A MARSHALL CAVENDISH 8 COMPUTER COURSE IN WEEKLY PARTS

INPUT

LEARN PROGRAMMING - FOR FUN AND THE FUTURE

UK £1.00
Republic of Ireland £1.25
Malta 85c
Australia $2.25
New Zealand $2.95



INPUT

Vol 1 No 8

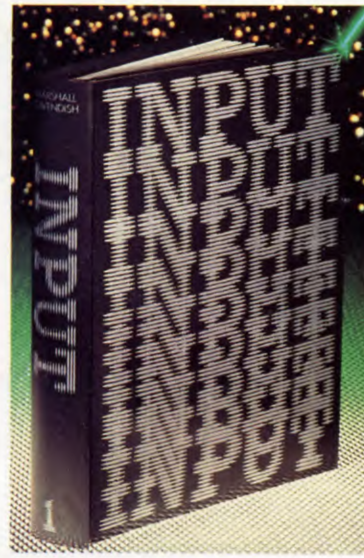
INDEX

The last part of INPUT, Part 52, will contain a complete, cross-referenced index. For easy access to your growing collection, a cumulative index to the contents of each issue is contained on the inside back cover.

PICTURE CREDITS

Front cover, Dave King. Pages 225, 226, 228, Nick Farmer. Pages 230, 232, 233, 234, 235, Tudor Art Studios. Pages 236, 237, 238, Chris Lyon. Pages 240, 242, 244, Dave King/Ian Craig/Roy Flukes. Pages 243, 245, 246, Ian Stephen. Page 248, Associated Press/Graham Bingham. Pages 250, 252, 254, JD Audio Visual. Page 255, Ray Duns. Equipment loaned by Lasky's, Tottenham Court Road, London W1.

Published by Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA, England. Printed by Artisan Presss, Leicester and Howard Hunt Litho, London.



There are four binders each holding 13 issues.

HOW TO ORDER YOUR BINDERS

UK and Republic of Ireland: Send £4.95 (inc p & p) (IR£5.45) for each binder to the address below:
Marshall Cavendish Services Ltd, Department 980, Newtown Road, Hove, Sussex BN3 7DN

Australia: See inserts for details, or write to INPUT, Gordon and Gotch Ltd, PO Box 213, Alexandria, NSW 2015

New Zealand: See inserts for details, or write to INPUT, Gordon and Gotch (NZ) Ltd, PO Box 1595, Wellington

Malta: Binders are available from local newsagents.

BACK NUMBERS

Copies of any part of INPUT can be obtained from the following addresses at the regular cover price, with no extra charge for postage and packing:

UK and Republic of Ireland:
INPUT, Dept AN, Marshall Cavendish Services, Newtown Road, Hove BN3 7DN

Australia, New Zealand and Malta:
Back numbers are available through your local newsagent

COPIES BY POST

Our Subscription Department can supply your copies direct to you regularly at £1.00 each. For example the cost of 26 issues is £26.00; for any other quantity simply multiply the number of issues required by £1.00. These rates apply anywhere in the world. Send your order, with payment to:
Subscription Department, Marshall Cavendish Services Ltd, Newtown Road, Hove, Sussex BN3 7DN
Please state the title of the publication and the part from which you wish to start.

HOW TO PAY: Readers in UK and Republic of Ireland: All cheques or postal orders for binders, back numbers and copies by post should be made payable to:
Marshall Cavendish Partworks Ltd.

QUERIES: When writing in, please give the make and model of your computer, as well as the Part No., page and line where the program is rejected or where it does not work. We can only answer specific queries – and please do not telephone. Send your queries to INPUT Queries, Marshall Cavendish Partworks Ltd, 58 Old Compton Street, London W1V 5PA.

INPUT IS SPECIALLY DESIGNED FOR:

The SINCLAIR ZX SPECTRUM (16K, 48K, 128 and +), COMMODORE 64 and 128, ACORN ELECTRON, BBC B and B+, and the DRAGON 32 and 64.

In addition, many of the programs and explanations are also suitable for the SINCLAIR ZX81, COMMODORE VIC 20, and TANDY COLOUR COMPUTER in 32K with extended BASIC. Programs and text which are specifically for particular machines are indicated by the following symbols:

SPECTRUM 16K, 48K, 128, and + **COMMODORE 64 and 128**

ACORN ELECTRON, BBC B and B+ **DRAGON 32 and 64**

ZX81 **VIC 20** **TANDY TRS80 COLOUR COMPUTER**

PRINTERS: THE CHOICES

- DO YOU NEED A PRINTER?
- CHOOSING THE RIGHT TYPE
- BUYING THE PAPER
- CONNECTING THE PRINTER TO YOUR COMPUTER

If you've ever wanted a 'hard' copy of your program or a screen dump of some graphics then you'll need a printer. But how do you know which sort to buy?

Buying a printer is one way to extend the usefulness of your computer and develop your programming skills. But it can be a very expensive investment, and since the choices and range of prices are just as varied as for computers, only the best-informed micro user can be sure of the need for a printer or of buying the most suitable one.

The normal way in which a computer outputs information to the user is onto a TV screen or VDU. A printer is just an alternative means of displaying the same information. But don't imagine that both devices have the same uses—one is no substitute for the other. The key role of a printer is to give a 'hard' copy or permanent record of any information that can be brought to the screen.

For example, there are many attractive graphic designs—computer art, maybe—that you could print and file or even frame, but without a printer they exist only fleetingly on the VDU screen. If you write a lengthy program, you may well want a record of the listing to keep or to work on away from the computer. If you write or buy business programs, you will need to print graphs or charts to be included in reports. And for Computer Aided Design you need a record of shapes or objects that can be referred to away from the computer.

All home computers have a keyboard, which is similar to that of a typewriter, so you can type and print correspondence, data—or even a book—as on a typewriter. Your capacity to handle text can be extended far beyond mere typing and into the realms of word processing, where you can manipulate text in



A single column of blunt needles strikes an inked ribbon to form the pattern of dots that make up each character of a dot matrix printer

all sorts of ways. Given the right software, your micro already has this ability, but a carefully composed block of text—whether it is just a short letter or even the contents of a booklet—is of little use if it cannot be printed.

And almost all users would find a printer helpful as a programming aid. As well as providing a back-up or copy of your work, it is often far easier to spot mistakes on paper than by scrolling through the program on a VDU screen. The merits of the VDU are many but, generally, people are more familiar with printed information so they find it easier to work with than an electronic image.

These attributes may well make a printer appear an essential aid to all computer users. But it is as well to remember that these abilities are not all built into every printer, and the most versatile printers are prohibitively expensive. For example, not all printers can handle graphics, and only a few can print in more than one colour—usually (although not always) at great expense. For this reason, it is most important to assess your needs and choose wisely. The type you choose depends on the application for which it is intended. To help you choose, start by thinking about the working characteristics of the different systems that are available.

TYPES OF PRINTER

There are four types of printer that might be suitable for the home micro user. These are the impact dot matrix, daisywheel, thermal, and pen printers.

The impact dot matrix printer is the most popular type. As with all modern printers, the carriage remains stationary while the print head advances across the width of the paper. The print head contains a closely grouped vertical column of needles which are attached to solenoids. When a solenoid is activated, its needle is pushed forwards against an ink ribbon onto the paper, making an ink dot. By firing the right combination of needles and advancing the position of the print head, a character is formed.

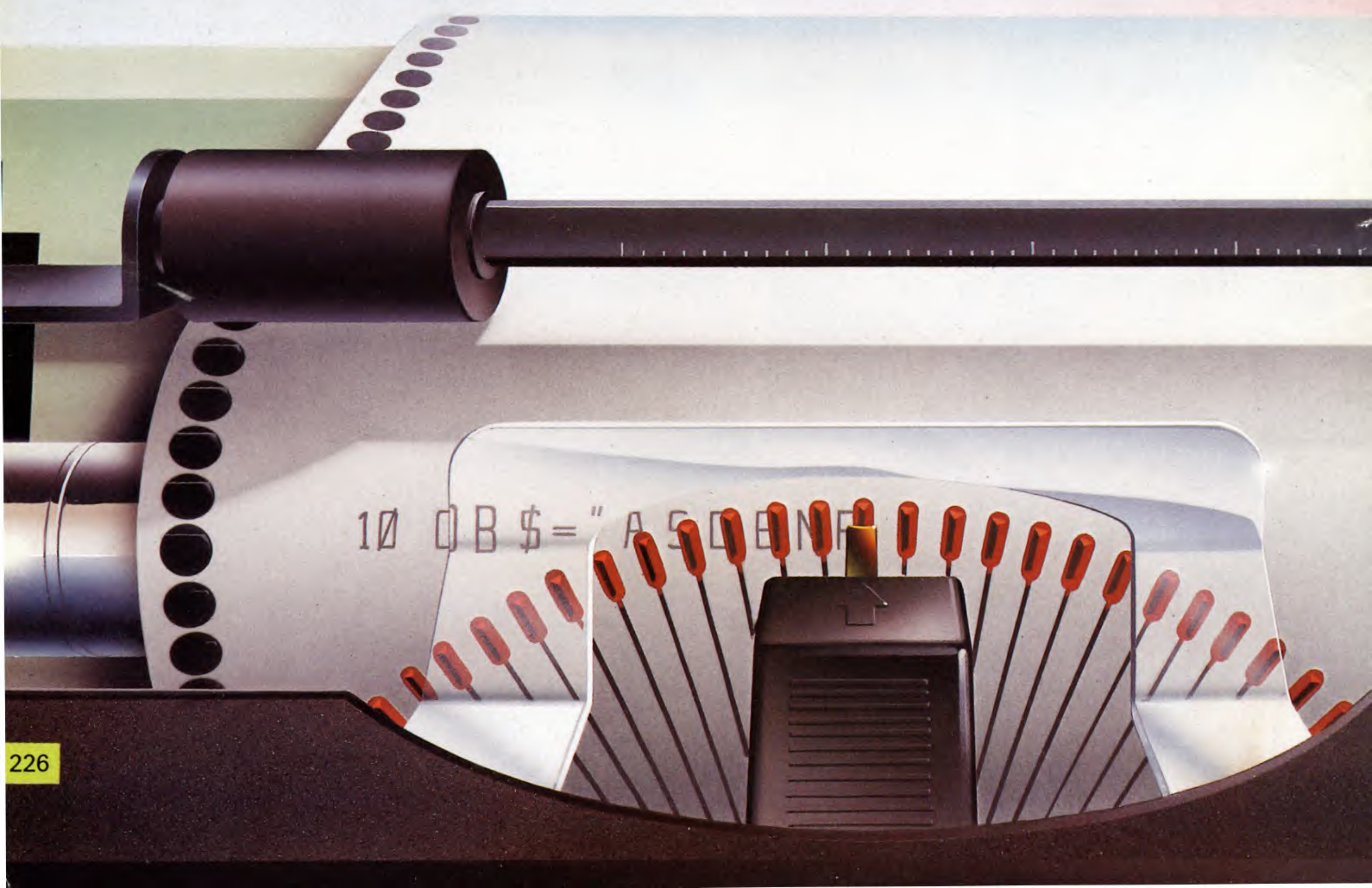
Commodore users have a choice of two dot matrix printers which are dedicated to their machines—the difference is in price and performance. The advantage of these is that they do not need interfacing—see page 229—and that they will print out Commodore ROM graphics characters. Other printers will not do this, which is a disadvantage for listing programs, for example.

Printing speeds of the dot matrix are among the fastest available, ranging from 80 characters per second (cps) to over 400. One disadvantage of some of these printers, however, is that the characters do not have true descenders—the 'tails' on letters, such as 'g' and 'q'. They make the entire character sit above the line, without the tail dropping below. True descenders are desirable for clarity and attractive presentation, and this feature is available on the better dot matrix printers.

Early designs of impact dot matrix printers used a seven-needle print head to produce characters in a matrix, or pattern, of 5×7 dots—five columns of seven rows. More recent designs employ nine needles to produce a 7×9 or 9×9 matrix.

Even with the best dot matrix printers, the characters can appear distinctly dotted, although they are clearly legible. They are quite suitable for most non-professional work, and can be made to print pictures from the screen. But for quality presentation, attempts have been made to improve the print clarity of the dot matrix printer.

One method, known as multipass printing, makes the print head print each line of text twice—once in one direction, and a second time in the opposite direction—before the paper is advanced. On the second pass, there is, invariably, a minute misalignment between



the printing needles and the characters already printed, so the needles tend to print in the spaces between dots. This effectively gives bolder, less-dotted characters, but of course it halves the printing speed.

Another method (which incurs no significant loss of printer speed) uses multiple columns of needles in the print head. The characters are formed in multiple steps—the right column fires first, then as it settles, the one to its left fires, and so on.

Although these methods give a significant improvement, the quality of dot matrix printers is below the best possible. Whenever high quality (such as formal correspondence) is required, the best choice is a solid-fount printer. Solid-fount means that the letters are formed by raised type characters which are an exact mirror image of the shape you want to print—just like the letters on an ordinary typewriter.

SOLID FOUNT PRINTERS

The first solid-fount printers for use with microcomputers were in fact adapted from golf-ball electric typewriters. Although the print quality was excellent, the mass of the golf-ball print head (designed for the fastest typist's speeds of up to 15 characters per second) slowed down the printers and made them unreliable. Consequently, the daisywheel printer (also known as the petal printer) was developed.

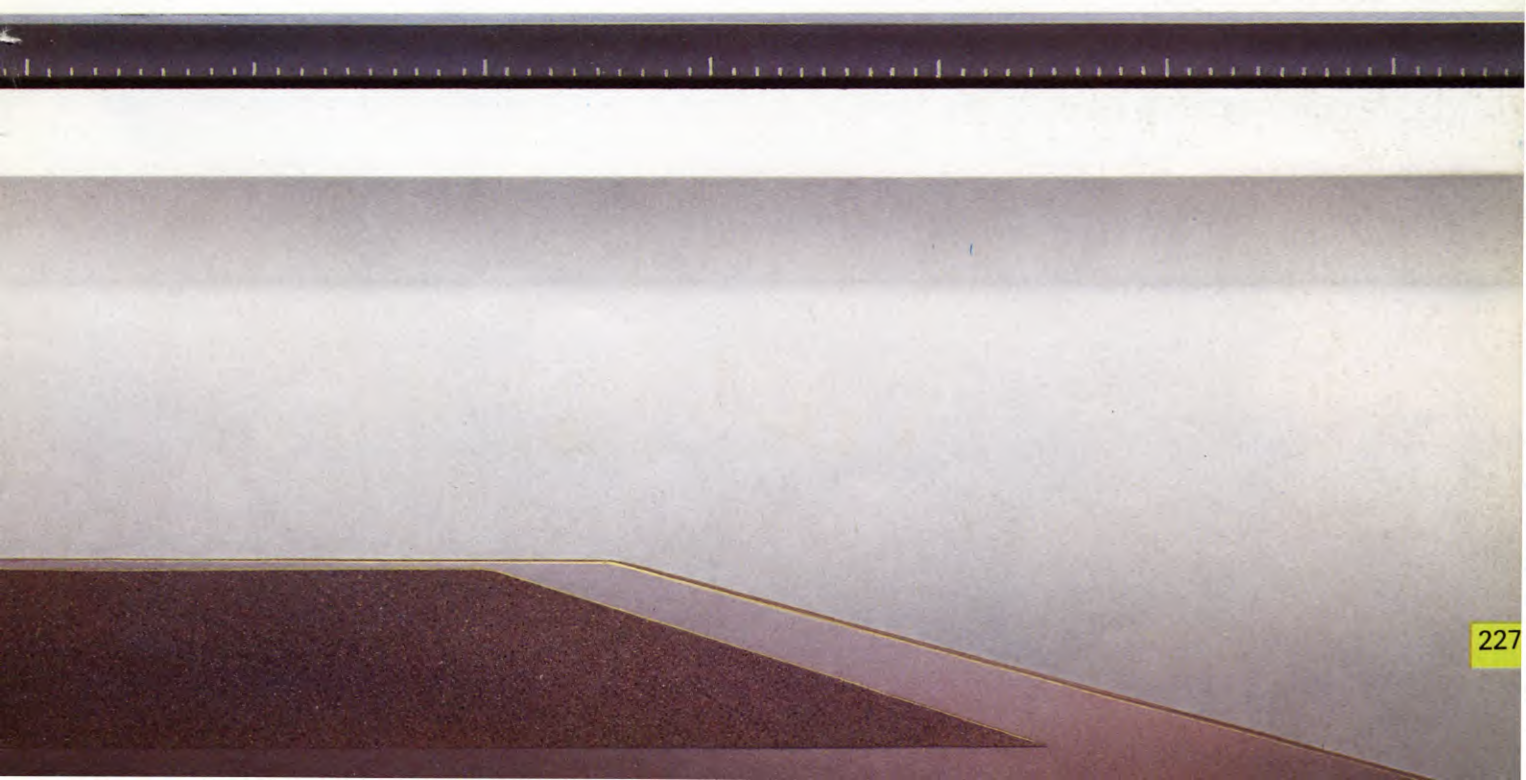
Daisywheel printers employ a removable printing element which resembles a daisy flower with petals radiating from the central cluster. Each petal has an embossed character at its tip, which is struck by a hammer when rotated into place. Because the daisywheel is removable, it is a simple matter to change to a different typeface, just by interchanging the wheels. The print quality is generally equal to, or better than, that of a typewriter and far superior to that of a multipass, impact dot-matrix printer, but printing speeds are slower than those of matrix printers. Speeds of between eight and 70 cps are usual.

To improve printing speed, many modern daisywheel printers are bi-directional, so that they print alternate lines from left to right and from right to left. This disposes of the need for carriage returns, which waste time. These printers also have circuits that tell the print head when it is quicker to return at the end of a short line to pick up the next line, or to continue to the edge of the paper, then print the next line in the opposite direction.

A number of text enhancements are possible with daisywheel printers. Bold type, underlining and true descenders are standard. Type styles can be combined by changing wheels between a first and second pass. Modern daisywheels are made of plastic, for lightness. The petals are made of soft plastic so that they are elastic and return to the starting position after being struck. The characters themselves are made of hard resin. They strike the ribbon with sharp edges so the print quality is clear, and the ribbon lasts a long time. There are three type sizes: four, five and six characters per centimetre. Replacement



Daisywheel printers are unsurpassed for high-quality printing. Changing the set of characters is as easy as replacing the daisywheel itself (shown above)



wheels cost about as much as a cassette copy of a good adventure game.

NON-IMPACT PRINTERS

Less popular than impact printers are the non-impact dot matrix types. The best known of these are the thermal and the electrostatic printers. Thermal printers require a paper coated with a heat-sensitive chemical. The paper is expensive but printing does not require a ribbon. The print head comprises a matrix of small heating elements. When the pattern to be printed is selected, the elements that make up the pattern are heated to between 100 and 150°C, darkening the heat-sensitive dyes on the paper.

One of the cheapest printers on the market is Sinclair's own ZX printer—a thermal type. This is compatible with both the Spectrum and the ZX81, and connects to the terminals in the machine's user port. It costs about a third as much as the Spectrum micro itself.

Naturally, such a modest sum won't buy a printer with the highest performance, but it is satisfactory for most needs that don't require letter-quality printing. It allows you to print text or graphics, and the typeface is readable though not always sharp. For providing a hard copy of a program listing or screen image, however, this is quite acceptable.

Besides its rock-bottom price, the ZX printer is remarkable for its smallness—it fits comfortably in the palm of one hand and weighs barely one kilogram. These dimensions limit the number of characters per line to 32—the same as the screen display. The paper is roll-fed and very narrow, only 100 mm (4 ins) wide and has a distinctive silvery sheen because of its aluminium coating.

If you are a ZX81 user then there is one great disadvantage: you need a more powerful power supply to drive the printer—which takes its power from the computer.

Electrostatic printers require a supply of liquid toner (usually carbon particles suspended in a solution of iso-paraffin). Characters are formed by charging the coated paper, which is then passed through a toner solution. The black particles in the toner adhere to the charged areas of the paper. The excess toner is removed and the paper is heated to fix it.

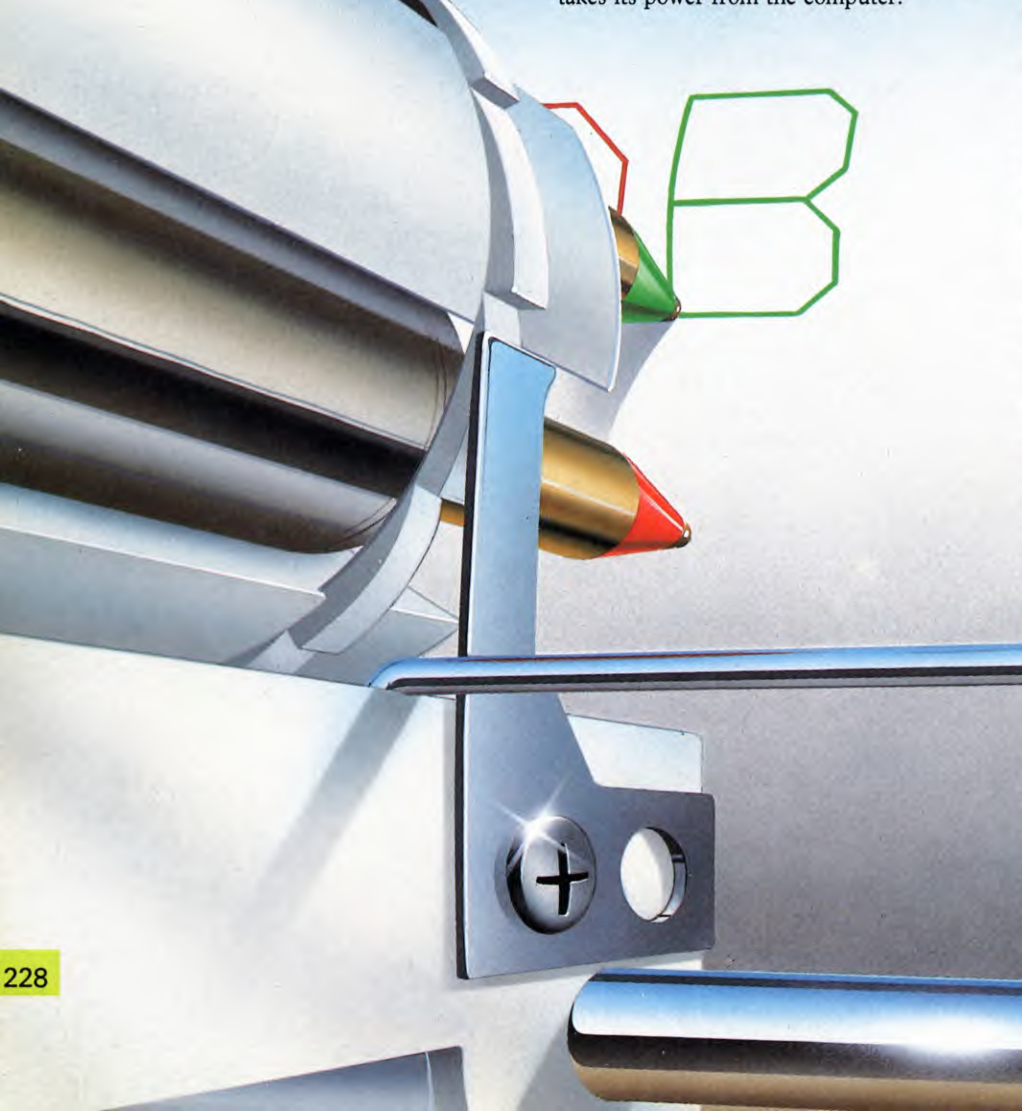
Thermal and electrostatic printers are more reliable than impact dot matrix types, (because there is less mechanical wear) but they have a much lower resolution.

One of the most interesting recent developments is a low-cost pen printer/plotter. This draws (with inked pens) pre-programmed characters on to the paper, but was designed specifically to produce charts and graphs. On less-expensive models, ball-point pens are used one at a time to draw on to ordinary untreated paper. Commodore users can buy a printer of this type which is dedicated to their machines.

Typically, the write head holds up to four pens, so multicolour prints can be drawn. The standard alphanumeric character set is neat and easily legible, but you could program your own characters using your computer's graphic facilities. Print speeds with the standard character set are about 12 cps. A replacement set of four pens costs about the same as a couple of data cassettes.

PAPER CONSIDERATIONS

When selecting a printer, it is most important to know how you want the printed product to look. This applies not only to the typeface but also the paper size and appearance. Besides the coated papers required by thermal and electrostatic printers, there is a wide variety of paper styles available in widths ranging from 10 to 40 cm. Some printers can use several different types and sizes of paper; others only one. Paper can be bought as standard loose sheets (just like normal typing paper), in rolls, or fanfolded. The fanfold type is the paper normally associated with computer printouts—it has sprocket holes along each side and



Pen printers have widespread applications in Computer Aided Design, and they are useful for generating a full range of alphanumeric characters

is supplied as a continuous sheet folded concertina-style with perforations between each 'page', so it can be separated into sheets or torn at convenient lengths.

Each type of paper requires its own feed mechanism. For fanfold paper a tractor or sprocket feed is used. The holes along each side of the paper mesh with sprocket pins on small wheels, which rotate to drive the paper past the print head. When used with an impact printer, multiple copies can be printed using carbon paper interleaved between the sheets.

A friction feed mechanism (similar to a typewriter feed) is required to print on loose sheets and roll paper. The paper is pressed against moving plates by pressure rollers. Roll paper is fed from an axle; a different input tray is necessary for loose sheets. For correspondence-quality printing, loose sheet is undoubtedly the best; fanfold and rolls are better for program printouts and graphics.

INTERFACING

Regardless of which type of printer you decide to buy, the most important point to consider is if it is compatible, or can be made compatible, with your make of computer. The computer can communicate with a printer only if the machines are suitably interfaced.

Essentially, an interface is the hardware (together with any software required to control it) that makes it possible to connect two systems. Some systems can be connected simply with a lead, either because they have been made compatible during manufacture or because they are sold with interfaces. Others are sold entirely without interface, or with only part of the interface required—just the hardware, maybe.

Do not wait until you have bought a printer to find out about interface requirements, because the additional cost can amount to as much as a third of the price of the machine.

If both machines are from the same manufacturer, they are usually either already compatible, or can be made so by adding an interface. And independent printer manufacturers try to make their printers compatible with as many computers as possible—usually by providing suitable interfaces. So although interfacing is crucial, it need not be a problem—if you make sure before you buy.

You can buy interface conversion modules for some machines, but there is also a possibility that the software you wish to run—particularly some word processing programs—may require a particular interface.

To ensure some degree of compatibility between the products of different manufacturers, a number of interface standards exist. Probably the most famous is the RS232C, which is a serial interface—data for each character is transmitted and received sequentially. Both the computer and the printer, however, deal with information for any one character in parallel (at the same time), so some means of conversion to and from a serial format is necessary.

These duties are left to an integrated circuit chip known as a Universal Asynchronous Receiver and Transmitter (UART). A UART in the computer receives coded characters either from the keyboard or memory, converts them from parallel into a serial format, adds other transmission codes and, providing the printer is ready to accept data, transmits the string of characters.

An identical UART in the printer accepts the data, decodes it and, so long as there are no errors, outputs the data in parallel format. Data transfer rates (Baud rates) vary between different printers from 75 to 19,200 bits per second, which is equivalent to about 7 to 1800 characters per second.

To overcome the speed limitations on the computing time caused by the slowness of the printer mechanism, a buffer memory is often employed. Information to be printed is stored in the buffer, releasing the computer to perform other tasks. Memory sizes range from 80 characters to 8000. The larger the memory, the less frequently data transmission needs take place and, consequently, the less of a demand the printer places on computer time. So a large memory is recommended.

The other type of serial interface commonly found on printers is the 20 mA current loop interface. Originally used on teletype terminals, it has been largely replaced by RS232 as it is not as fast and so cannot be used for the quicker, modern printers.

The most popular printer interface, however, is the Centronics interface. Designed by the printer manufacturer of the same name in the 1970s, it has become the accepted standard, mainly because of its simplicity. Centronics is a parallel interface, so data is transferred simultaneously over a set of parallel data lines.

Originally developed for interfacing measurement instruments with a computer, IEEE 488 is a sophisticated parallel interface standard allowing fully duplex (simultaneous two-way) communications. Consequently, because of its availability on many microcomputers, it has been incorporated into a number of printer designs. Generally, however, many of the facilities it offers are not necessary for use with printers, and Centronics offer a better parallel interface option.

By the time you have decided on the type of printer to buy, you should have some idea of the price—including interface. Of course, price may have been the deciding factor in the first place. Be warned that some machines are sold without a connecting lead, which can cost twice as much as a low-price game on disk.

Whether you opt for a printer that costs as much as your computer or one costing ten times as much, you are well advised to see the particular software package and printer you want working connected to your computer before you commit yourself. Usually, demonstration units are available, but might not be displayed, so don't be afraid to ask.



BREAKING THE SOUND BARRIER

Liven up your games and make them more exciting by adding some sound effects—anything from beeps, explosions and alien-zapping noises to a funeral march or a steam train

Games programs usually have all kinds of sound effects to make them more exciting—explosions, zaps, pings, small tunes, or whatever takes the programmer's fancy.

This part of Games Programming aims to provide you with a small library of ready-made sound effects. You can use the effects as they are or as a basis for your own experiments.

Remember that there are no hard-and-fast rules for producing sound effects. If your game needs a noise to accompany a figure being assaulted with a sockful of lumpy custard, you'll have to sit at your machine and experiment. Conversely, some seemingly-unlikely sounds can be wonderful if you happen to invent the right kind of graphics.

How you incorporate sound effects into your programs depends on their complexity and how often they are used. For simple effects that are only used once in a program, it's best to put them after an IF ... THEN, but for more complex ones, or ones used many times, it's best to use them in a subroutine.

The sophistication of the effects you will be able to achieve depends partly upon your own programming skill, but it is also determined by the sound capabilities of your computer. For example, the Spectrum's sound generator is limited to a BEEP whose pitch and duration can be controlled by the user. On the other hand, the Commodore and Acorn machines have very sophisticated sound generators capable of synthesizing a huge range of different effects. The ZX81 has no sound at all.

The Spectrum has a single sound facility, BEEP in BASIC. It's simple to program and can be used to liven up your programs considerably. For example, here is a routine which will play a series of bleeps:

```
8000 FOR n=1 TO 12
8010 BEEP .03,30
8020 NEXT n
```

As you can see from the program lines above, BEEP is followed by two numbers separated by a comma.

The first number determines the length of the note—the larger the number the longer the note. A value of one will cause a note one second long to be played, and larger numbers or decimal fractions work accordingly.

The second number sets the pitch of the note, Ø being middle C. Each whole number above or below this represents a semitone higher or lower—a step on to the next note on a piano keyboard, if you like. For sounds in games, very low pitches of about −32 are often used to imitate explosions and zaps.

The BEEP command will be explained in more detail in a future issue of INPUT, in particular how to use it for making music.

In the routine above, a FOR ... NEXT loop is used to play a series of notes. The lines below use two loops, with the control variable of each loop setting the pitch of each note. This results



- INVENTING THE RIGHT SOUND
- ADDING SOUND EFFECTS TO THE RANDOM MAZE GAME
- USING **BEEP**, **PLAY**, **SOUND** AND **ENVELOPE**
- MAKING EXPLOSIONS, PINGS AND ZAPPING NOISES
- FROM NOISE TO MUSICAL NOTES AND TUNES
- FURTHER EXPLOSIONS

in an effect which you might like to use when an alien has been zapped in your games.

```
8ØØØ FOR n=4 TO Ø STEP -1
8Ø1Ø BEEP .Ø1,n
8Ø2Ø NEXT n
8Ø3Ø FOR n=1 TO 4
8Ø4Ø BEEP .Ø1,n
8Ø5Ø NEXT n
```

Here is a routine using a FOR . . . NEXT loop in a similar way, but this time the effect is more of a 'congratulations' sound. You might use it when the player has killed all the monsters, or to celebrate 'hitting the jackpot' in a fruit machine game.

```
8ØØØ FOR n=1Ø TO 6Ø STEP 5
8Ø1Ø BEEP .Ø1,n
8Ø15 BEEP .Ø1,n-2
8Ø2Ø NEXT n
```

Try experimenting with the BEEP command to produce other sound effects for your games. Try using loops and subroutines as well, to liven up your programs.

SOUNDS FOR THE MAZE

The random maze game given in the last issue can be improved no end by the inclusion of some sound effects. Add Lines 365 and 5ØØ, to the original program, and change Line 4ØØ:

```
365 BEEP .Ø1,1Ø
4ØØ LET lives=lives-1:RESTORE 5ØØ:
    FOR f=Ø TO 1Ø: READ a,b: BEEPa,b:
    NEXT f: IF lives>Ø THEN GOTO 26Ø
5ØØ DATA .45,Ø,.3,Ø,.15,Ø,.45,Ø,.3,3,
    .15,2,.3,2,.15,Ø,.3,Ø,.15,-1,.45,Ø
```

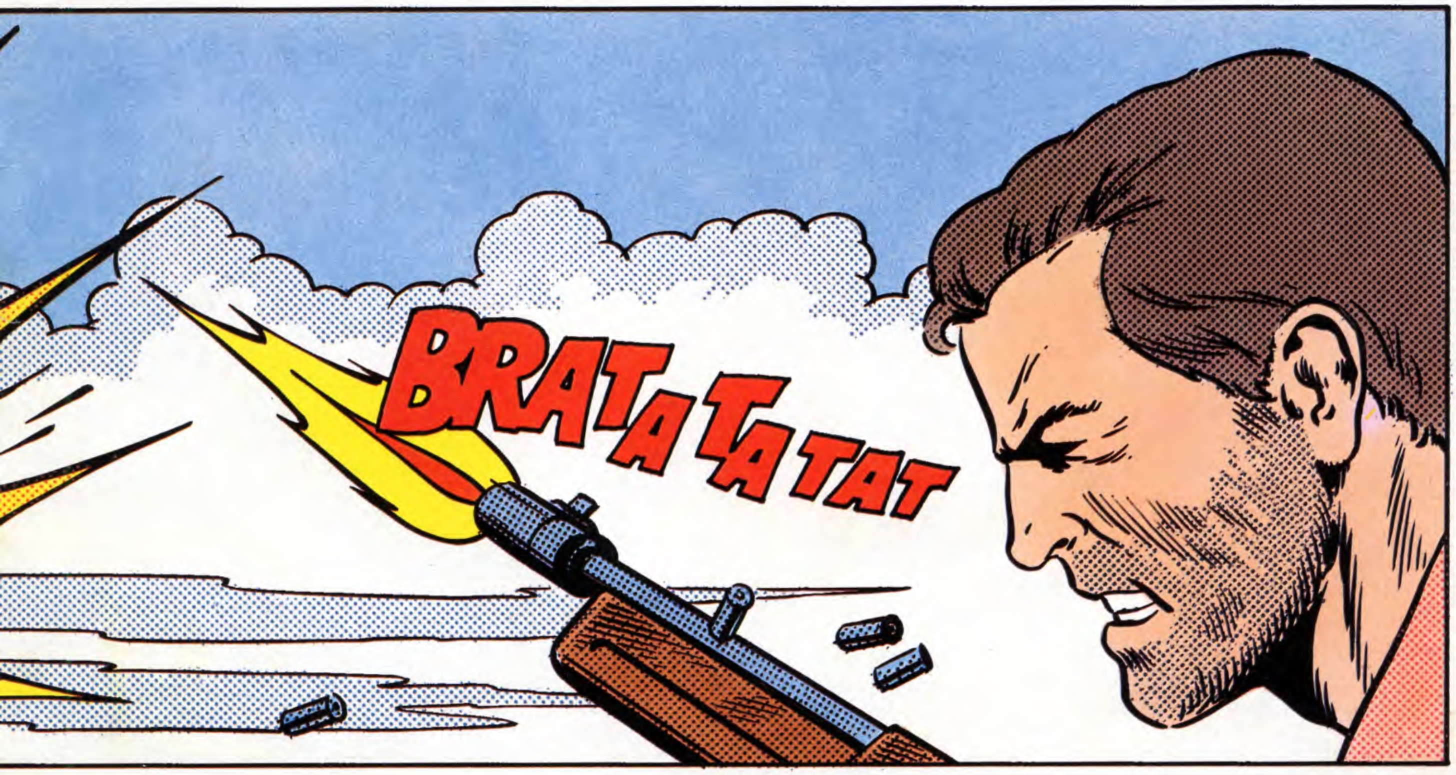
If you now RUN the program you will find that you are greeted by a sound whenever you reach the treasure, and by a funeral march whenever you lose a life. Try changing the values of a and b to create different effects.

This example shows that you will need to be careful when using sound effects in games, since the computer stops doing whatever it is doing while it is BEEPing, and so your game might contain some unwelcome pauses. Even if you only include short effects, this may render your game too slow to be playable.

The Commodore 64 has an extremely sophisticated sound feature employing three highly controllable 'voices' or sound channels. A wide variety of sounds are possible, starting with simple sound effects of the type that can be incorporated within any games program.

One, two or three voices can be used to create these effects and shortly you'll see how these can be combined. The 'nuts and bolts' explanation of how to program the SID chip is left to a later BASIC Programming article, however.

The SID chip used by the 64 is extremely powerful and a wide variety of interesting—and useful—sound effects can be created with comparatively little programming. Unlike the VIC graphics chip, which requires a large number of POKEs at almost every stage of the process, many of the necessary POKEs for the SID chip locations need only be made once. Thereafter, one or two subtle value changes can often radically alter the nature of the sound produced.



Up to five types of setting usually have to be made for each sound, and to explain the effects of these, key in this short program:

```
1Ø S=54272
3Ø POKE S+1,255
4Ø POKE S+5,219
5Ø POKE S+24,15
6Ø POKE S+4,129
7Ø FOR Z=1 TO 5ØØØ: NEXT Z
8Ø POKE S+4,128
1ØØ GOTO 3Ø
```

RUN the program and a few seconds later you should hear the sound of waves breaking on the shore. Note how the sound chip locations are represented in relation to the starting address (S) at 54272—this makes it much easier to remember which one you're dealing with. The first setting, S+1, is one of two possible *frequency* control registers—we're using the one which sets the high byte, which in effect means a frequency number of 255 times the figure used, the maximum figure 255 in this instance. Try changing this to something smaller, say 1Ø, and reRUN the program for a slightly different wave noise.

Location S+5 is the *envelope generator* which we need not bother too much about here. S+24 in the next line is the *volume setting* location when values of up to 15 are set. This is the maximum loudness, and it's worth setting this value and making any necessary adjustments on the volume control of the TV for the simple demonstrations here.

On the next line, S+4 is used. This is the location of the voice-1 control register which is used to select, amongst other things, the *waveform* (by setting one of the higher bit values). The value 129 selects a random noise waveform—try changing this to 33 or 17 for a high pitched sound. Leave the value at 17 and change the POKEd value in Line 3Ø to 1Ø and shorten the delay loop in Line 7Ø by putting in 5Ø in place of 5ØØØ. Now reRUN the program to hear how seascape sounds have been converted to that of a puffing steam train!

In a complete games program, such changes could be introduced by changing the values of variables at appropriate points and thus a wide variety of effects may be possible from a simple sequence of settings in which only a few values will need to be changed.

SOUNDS FOR THE MAZE

If you have SAVEd the maze game which appeared on page 196, you may like to add the following three lines once the program has been LOADed. You need to turn the Commodore off and on before LOADing:

```
1Ø2 S=54272:POKE S+5,33:POKE
   S+6,255:POKE S+24,15:POKE
   S+4,33:POKE S+1,Ø
2ØØ2 POKE S+1,Z
3ØØ5 POKE S+1,25Ø:FOR Z=1 TO
   1Ø:NEXT Z:POKE S+1,Ø
```

The first of these initializes the sound system, and the second and third add rather welcome sound effects which really do make the game much more interesting—see what you think!

Here are some other sound effects which you may like to experiment with. First, we have to set up the sound system: (NEW your computer first)

```
5 S=54272:W(1)=17:W(2)=33:
   W(3)=129
1Ø FOR Z=S TO S+24: POKE Z,Ø:
   NEXT Z
15 P=1 :W=1 :REM P (1-13)
   W (1-3)
2Ø POKE S+24, 15 :REM VOLUME
25 POKE S+5,15 :REM ATTACK/
   DECAY
3Ø POKE S+4,W(W):REM WAVEFORM
35 POKE S+6,15 :REM SUSTAIN/
   RELEASE
4Ø ON P GOSUB 55,6Ø,65,7Ø,75,8Ø,
   85,9Ø,95,1ØØ,1Ø5,11Ø,115
5Ø POKE S+4,W(W)-1:POKE S+5,Ø:
   POKE S,Ø:POKE S+1,Ø:END
55 FOR Z=1 TO 75 STEP.1: POKE
   S+1,Z: NEXT Z: RETURN
6Ø FOR Z=1 TO 75 STEP.1: POKE
   S+1,ABS(SIN(Z)*15):
   NEXT Z: RETURN
65 FOR Z=75 TO 5 STEP-1:POKE
   S+1,Z:POKE S,Z:NEXT Z:
   RETURN
7Ø FOR Z=1 TO 1ØØ:POKE S+1,
   RND(1)*75:NEXT Z:RETURN
75 FOR Z=1 TO 2ØØ:POKE S+1,ABS
   (TAN(Z)+5):NEXT Z:
   RETURN
8Ø POKE S+1,1Ø:POKE S,127:FOR
   Z=1 TO 15 STEP.Ø5:POKE
   S+24,Z:NEXT Z:RETURN
85 FOR Z=1 TO 25Ø STEP 1:POKE
   S+1,Z:POKE S+1,255-Z:
   NEXT Z:RETURN
9Ø FOR Z=5 TO 1ØØSTEP 5: FOR ZZ
   =1Ø TO 1ØØ STEP 1Ø:POKE S+1,
   Z:POKE S,ZZ:NEXT ZZ,Z:
   RETURN
95 FOR Z=2Ø TO 2ØØ STEP 1Ø: FOR
   ZZ=1 TO 2Ø:POKE S+1,Z-
   ZZ:POKE S+1,ZZ+5Ø:NEXT ZZ,Z:
   RETURN
1ØØ FOR Z=1 TO 4Ø:FOR ZZ=1 TO
   RND(1)*5Ø:POKE S+1,ZZ:
   NEXT ZZ,Z:RETURN
1Ø5 FOR Z=1Ø TO 1ØØ STEP 1Ø:
   FOR ZZ=5 TO Z:POKE S+1,
   Z-ZZ:NEXT ZZ,Z:RETURN
11Ø POKE S+1,RND(1)*2ØØ+5:FOR
   Z=1 TO RND(1)*5ØØ+1ØØ:
   NEXT Z:RETURN
115 POKE S+1,RND(1)*2ØØ+5:FOR
   Z=1 TO RND(1)*1ØØ STEP
   .2:POKE S+24,Z:NEXT Z:RETURN
```

Lines 1Ø to 5Ø initialize the system, and each line that follows is a separate sound effect which you can introduce into the program by an appropriate adjustment to the values P and W in Line 15. P represents the one-line sound effects of Lines 55 to 115. W is the waveform and adjusting the value of this from 1 to 3 selects *triangle*, *sawtooth* or *noise* forms—whichever is suitable for your effects.

RUNning the program in its entered form produces a loud howl. CLR the screen and LIST Line 15 and RUN the program again. You should now have Line 15 with RUN displayed below it on the screen, and it's a simple matter to make changes to both P and W to try out three variations of each sound effect line.

With P=2 W=1 you have, for instance, a warbling sound This changes to a Tardis-like sound when W=3. Other interesting effects are possible with the following combinations of P and W: P=7 W=1, P=9 W=1, P=1Ø W=1 or 2, and P=12 W=1. But there are plenty of others—try experimenting, you might be surprised at the results!

Further effects are possible by adjusting the POKEd value in Lines 25 and 35.

Sound on the Vic 20 works in a similar fashion to sound on the 64, but is a little less sophisticated.



Five special registers govern sound generation—36874 controls low notes, 36875 controls middle notes, and 36876 controls high notes; 36877 is a noise generator; 36878 controls the volume. All the registers normally contain the value Ø—off. The tone registers can contain values from 128 to 255, and the volume register can contain values from 1 to 15. If a number greater than Ø is POKEd into any of the tone registers they will continue generating sound until they are switched off by POKEing Ø back into them.

Here is a short library of sounds which will illustrate how the registers can be used to produce special effects. First an 'end of game' sound:

```
1Ø POKE 36878,15
2Ø FOR Z=128 TO 255:POKE
   36875,Z:NEXT Z
3Ø POKE 36875,Ø
```

or a motorbike:

```
1Ø POKE 36878,15:FOR Z=1 TO 1ØØ
2Ø POKE 36875,(15Ø+RND(1)*5)
3Ø POKE 36875,Ø:NEXT Z
```

Or this interesting sci-fi effect, borrowing values from other memory locations:

```
1Ø POKE 36878,15
2Ø FOR Z=6ØØØØ TO 6Ø999: POKE
   36876,PEEK(Z):NEXT
3Ø POKE 36876,Ø
```

The Acorn machines have two commands which can be used to produce sounds—SOUND and ENVELOPE. SOUND can be used on its own to produce simple effect, whilst ENVELOPE allows you to make modifications—quite literally 'shaping' the sound—so that more sophisticated effects can be obtained.

Here is the kind of effect you can get from using SOUND alone:

```
SOUND 1, -15,8Ø,1ØØ
```

The four numbers allow you to select channel, volume, pitch and note duration. You can choose either channel 1 or channel Ø—channel 1 allows you to produce pure musical tones and Ø allows you to produce noise.

In the volume position, −15 turns the sound on, and Ø turns the sound off. Numbers between 1 and 16 are ENVELOPE numbers, but more about that later. Note that earlier Acorn operating systems only allow ENVELOPES between 1 and 4.

Pitch may take numbers from Ø to 255. Ø produces a very low bass note, whilst 255 produces a high treble note.

Finally, you can vary duration between Ø and 255—the larger the number, the greater the duration. 255 will give you a continuous note—one which will play until you stop it, either by pressing ESCAPE or from within a program.

One of the best features of the sound on the Acorn machines is that the computer is free to do other things whilst it generates sounds. On some other micros everything stops when they make sounds—they can either make sounds *or* move graphics in an arcade-type game, for example.

Everything above applies to both the BBC and the Electron. However, there are some differences between the machines—the BBC has more sophisticated sound facilities.

Firstly, there are two extra sound channels, numbered 2 and 3. These are identical to channel 1 and are used for producing musical notes—so on this machine you can play two or three notes at the same time. Secondly, the volume may be varied in steps between Ø and −15. The Electron will take intermediate values, but any number less than Ø will be treated as 'on'.

SOUNDS FOR THE MAZE

If you SAVEd the random maze game from Games Programming 7 (page 197) you might like to add some sound effects.

Firstly, a small bleep which will sound as the man is moved around the maze:

```
795 SOUND1,-15,15Ø,1
```

The second sound happens when the player succeeds in recovering some treasure. Add Line 225 and change Line 83Ø:

```
225 ENVELOPE1,128,2,Ø,Ø,5Ø,Ø,Ø,127,
    Ø,Ø,-127,126,Ø
83Ø IF H=226 THEN SOUND 1,1,15Ø,1Ø:
    PRINT TAB(RND(36+D*4Ø)+1,
    RND(22)+4)CHR$(226): SC=SC
    +MT-TIME: TIME=Ø
```

And lastly, a funeral march for when the player loses a life:

```
891 RESTORE: FOR T=1 TO 11
892 READ D,P
893 SOUND1,-15,P,D
894 SOUND1,Ø,Ø,1
895 NEXT
896 DATA 12,5,8,5,4,5,12,5,8,17,4,13,
    8,13,4,5,8,5,4,1,12,5
```

ENVELOPE

The treasure-finding sound uses ENVELOPE as well as SOUND. ENVELOPE cannot be used on



its own because all it does is to modify the SOUND.

Unfortunately, there isn't enough space here to explain the complexities of ENVELOPE—there are 14 separate numbers associated with the command—but it will be fully explained in a later issue of INPUT. But some feel for what ENVELOPE does can be gained from just experimenting with the ENVELOPEs given in this issue.

If you do decide to use ENVELOPE as well as SOUND, the volume number in the SOUND command will have to match the ENVELOPE number. In other words, make sure that the second number in SOUND matches the first number in ENVELOPE.

MORE SOUND EFFECTS

Here are a variety of sound effects which you can put in your own games programs. The ENVELOPE line can be put anywhere in the program as long as it precedes the SOUND command which refers to it.

One of the most useful sound effects in games is an explosion like this one:

```
1Ø ENVELOPE1,131,Ø,Ø,Ø,Ø,Ø,Ø,126,
   −3,Ø,Ø,126,Ø
2Ø SOUND Ø,1,6,1ØØ
```

Alternatively, you may want a crash sound like this:

```
1Ø ENVELOPE1,1,Ø,Ø,Ø,Ø,Ø,Ø,126,
   −1,−3,−126,126,Ø
2Ø SOUND Ø,1,6Ø,35
```

Or a siren like this:

```
1Ø ENVELOPE3,1,3,−3,3,1Ø,2Ø,1Ø,
   127,Ø,Ø,Ø,126,126
2Ø SOUND 1,3,1ØØ,2Ø
```

These sounds would be most at home in a down-to-earth game—say, one involving tanks, aircraft, or even in a car racing game.

For an alien-zapping game you'll need some more abstract sounds like this one:

```
1Ø ENVELOPE1,13Ø,2Ø,−2,Ø,5,3Ø,Ø,
   126,Ø,Ø,−7,126,Ø
2Ø SOUND 1,1,1ØØ,1Ø
```

Or a ping:

```
1Ø ENVELOPE2,1,Ø,Ø,Ø,Ø,Ø,Ø,126,
   −1,−1,−1,126,Ø
2Ø SOUND 1,2,17Ø,3
```

Or a vibro beam:

```
1Ø ENVELOPE1,1,−16,48,−16,3,1,
   3,7,65,Ø,−4,65,126
2Ø SOUND 1,1,22Ø,8
```

You can use these effects as they stand, or you could try matching a different ENVELOPE with the SOUND. All you need remember to do is to match the ENVELOPE numbers. You may also wish to try altering some of the numbers in SOUND and ENVELOPE. It's worth spending some time experimenting.

There are two keywords on the Dragon and Tandy which produce sound effects—SOUND and PLAY. Both use the same sound generator, and you can't really affect the quality of the note it produces. SOUND controls the pitch and duration of the note, while PLAY allows you to select a string of notes—to play a tune, for example. SOUND is the simplest to use and can be used to produce effects like the one in the following program. When you RUN it don't forget to turn up the volume on your TV, or you won't hear a thing.

```
1Ø FOR Q=1 TO 5
2Ø SOUND 2ØØ,1
3Ø NEXT Q
```

The series of blips can be altered by changing the numbers after the SOUND command. The first number controls the pitch of the note, and can take a value from 1 to 255. The lowest note is produced by 1, and the highest by 255—as a guide for those of you who know something about music, 89 gives middle C.

The second figure after SOUND regulates the length of the blips. Again, the range of values is from 1 to 255—16 gives about one second's duration.

These blips would be very suitable for use in a space or fantasy game, but wouldn't be much use with a more realistic type of theme. Just like the screen flash on page 161, you have to be careful when to use them. Always use appropriate sound effects.

If you decide that you need a more sophisticated type of effect you'll have to abandon the SOUND command because it is too limited. Instead you'll have to use PLAY as explained later.

SOUNDS FOR THE MAZE

If you have SAVEd the random maze game on pages 194 to 196 you can greatly improve the game by adding these sound effects.

You should change Line 23Ø so that it reads:

```
23Ø IF X<>LX OR Y<>LY THENPUT
   (X1,Y1)−(X1+BS−1,Y1+BS−1),B,
   PSET:LX=X:LY=Y:PLAY"T5ØO5DG"
```

and change Line 24Ø to:

```
24Ø IFF=1 THENF=Ø:SC=SC+
   (TI−TIMER):TI=TI−1Ø:PLAY
```





```
"T1ØO3BCDEFGA":GOTO13Ø
```

and finally, change Line 5ØØ to:

```
5ØØ CLS:SCREENØ,Ø:PLAY"T3O2
    L2CL4CL12CL2CL4D#L8DL4DL8
    CL4CO1L8BO2L2C" :LI=LI-1
```

You can use the computer's editor, or re-type the lines in full.

Now RUN the program and listen to what happens as you move the man around. You'll get a different sound when the treasure is recovered, and a funeral march when a life is lost.

One thing that you must be careful to avoid is that your sound effects do not slow down your game too much. Every time the computer makes a sound it stops doing everything else. In games programming, this means that any screen movement stops whilst the sound is being made—in fact, when you want a pause, you can use a sound instead of a FOR . . . NEXT loop to create a delay in your programs. For this reason, the sound in Line 23Ø is very short so that the delay is kept to a minimum. You can even 'fine tune' your programs with sound.

THE PLAY COMMAND

PLAY always operates on a string containing instructions for the computer. The string in Line 23Ø of the previous program contains the instructions T5ØO5DG. T stands for Tempo—speed, if you like—and can be set to any value from 1 to 255. O is the Octave, and may take a value from 1 to 5—1 is the lowest and 5 the highest. The last two letters, D and G, are notes. Very broadly, what T5ØO5DG means is 'Play two high notes fairly quickly'.

Try altering the values of T and O to see what kind of effects you can get.

In Line 24Ø the string is T1ØO3BCDEFGA, which plays a rising scale, except for the first note.

The funeral march in Line 5ØØ is more complex. A full explanation of how to write music on the computer will have to wait until later, but notice how the note length, L, is varied during the tune. In fact you can choose to vary any of the parameters in the string at any point during the tune, making the PLAY command very flexible.

EXPLOSIONS

The PLAY command can also be used to produce sound effects for explosions which can be very useful in games.

The following sound effects can be used either on their own, or with the visual effects you saw on pages 161 to 167.

Try this effect:

```
1Ø PLAY "T16ØO1L3ØGF#FE#D#DC#
    D#DFE#"
```

You could use it when something is hit, or explodes on the screen. For example, you could incorporate it in the program on page 100, by adding the PLAY command to Line 27Ø before the GOTO.

A nice reverberating sound can be set up like this:

```
1Ø PLAY "T8ØO1L2BFBFBFBFBFBFBFBF"
```

Two notes are repeated over and over in the lowest octave. Try manipulating the tempo and note length to get variations on this sound.

You can get your sound effects to repeat by using a routine like this:

```
1Ø F$="T1ØØO2L1ØAGBE#DFADF#"
2Ø F$=F$+F$
3Ø PLAY F$
```

Line 1Ø is the string of instructions for the sound. Line 2Ø concatenates the string so that when Line 3Ø PLAYs it, the sound repeats.

A SIREN

Being able to make a sound repeat has other applications. Suppose you want a siren to warn the player of approaching enemies. You could use a program like this:

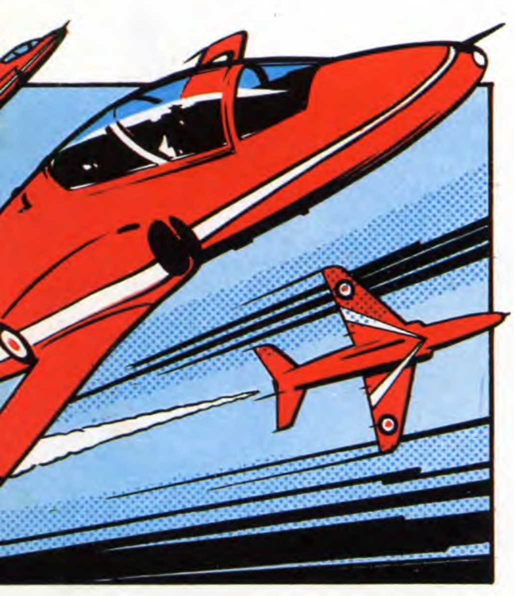
```
1Ø CLEAR 25Ø
2Ø F$="T15ØO4L8CDD#EF#FGF#
    FED#DC#CD#DC#C"
3Ø F$=F$+F$+F$+F$
4Ø PLAY F$
```

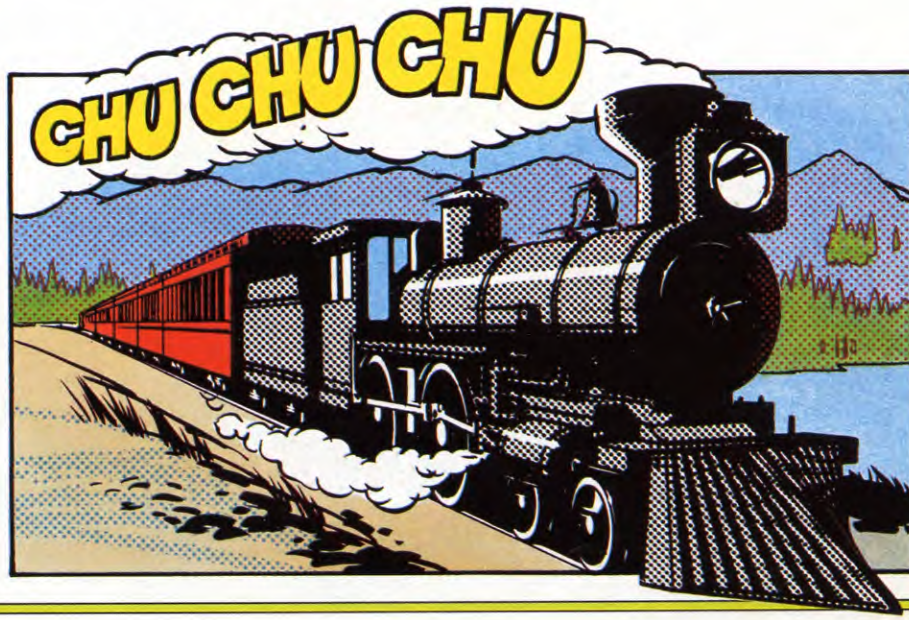
With the siren effect a very long string will be built up by concatenating four lots of the original F$ before it is played. The Dragon and Tandy have only a limited amount of string storage when the machines are first switched on, and the new sound effect will be much too large for the available memory.

You therefore have to reserve more string space by CLEARing some more memory—25Ø bytes of string space are reserved by Line 1Ø.

Be careful that you reserve 25Ø extra bytes of memory when you use the sound effect in a games program, or else you will suffer an OS–out of string space–error.

Don't be disappointed if your experiments do not produce effects that are comparable to the best effects in commercial games. Some very good effects are possible from BASIC, but the most sophisticated are written in machine code—a topic which will be dealt with later on in INPUT.





GETTING TO THE HEART OF IT

To understand machine code you have to understand how the microprocessor works and what it can do. It is the brain of the machine, as well as its heart

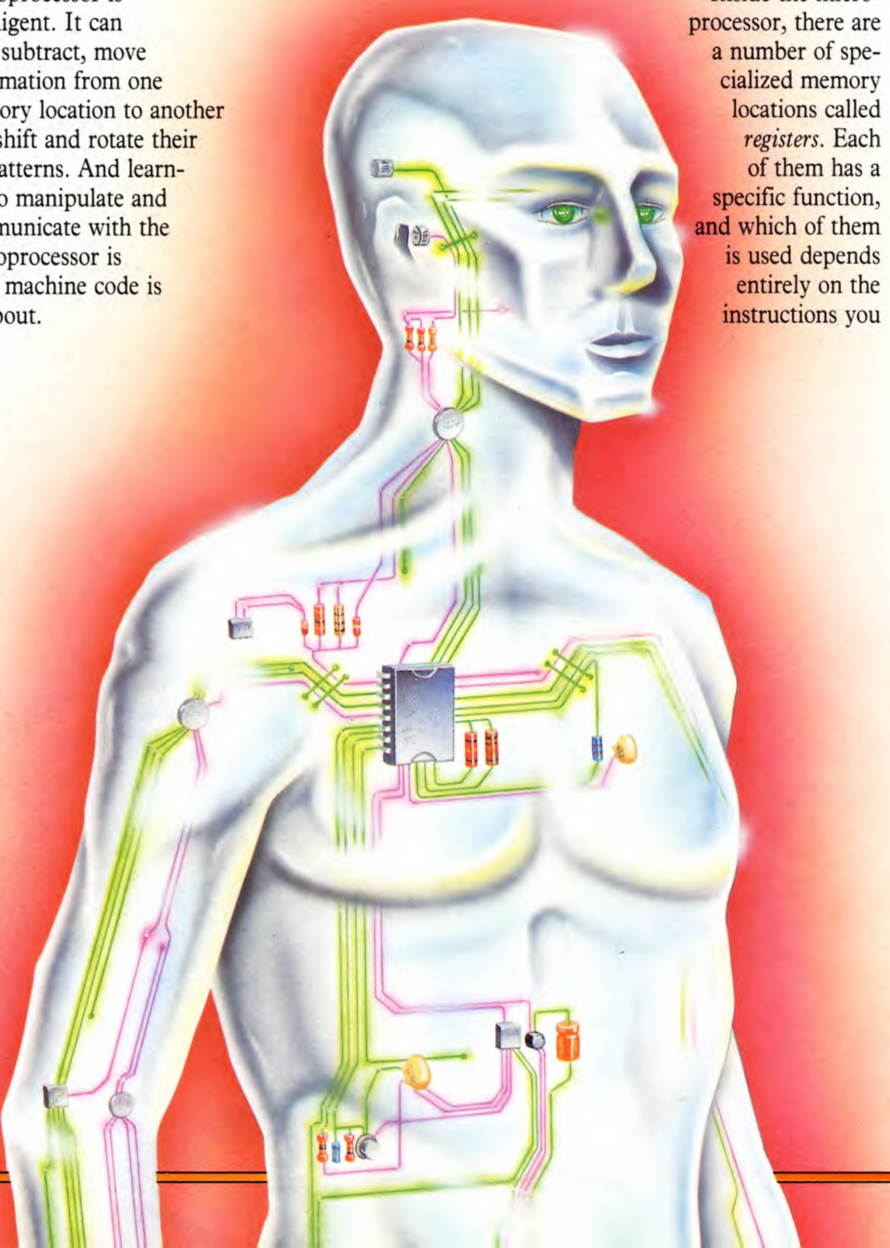
Computers have memories. But they do a great deal more than just remember things, because they are also able to manipulate what they memorize. The manipulation is done by the Central Processing Unit (CPU), also called the microprocessor.

A microprocessor looks like any other chip but it is very different from the others. They are passive devices that simply remember what they are told to. But the microprocessor is intelligent. It can add, subtract, move information from one memory location to another and shift and rotate their bit patterns. And learning to manipulate and communicate with the microprocessor is what machine code is all about.

THE HEART OF THE MACHINE

Everything that happens in your computer is under the direct control of the microprocessor. It's the heart of the machine. It does the arithmetic, copies data from one memory location to another, assigns functions to areas of memory and runs your programs. And when you write machine code programs, it is the microprocessor you are communicating with.

Inside the microprocessor, there are a number of specialized memory locations called *registers*. Each of them has a specific function, and which of them is used depends entirely on the instructions you give when you write your machine code.



One of the important things about registers is that they don't have addresses. This makes them very fast to use.

THE REGISTERS

Each of the various chips in the home computers under consideration here has a different set of registers. But some of the registers are still common to all the machines. All of them have a 16-bit PC register. This is the *program counter*. It stores the address of the next byte of the machine code program to be executed. In other words it tells the computer whereabouts it is in the program. And as each byte of the machine code program is executed, the PC register is incremented by 1.

All home computers have at least one eight-bit *accumulator* or A register. These are where the computer does its arithmetic. If you want to add one number to another, or subtract one number from another, one of the numbers should be in an accumulator. This also affects multiplication and division—which are only combinations of logical operations with additions and subtractions.

Each of these machines also has a couple of *index registers*. Those in the Spectrum, the ZX81, the Tandy and the Dragon are 16-bit registers which contain addresses. These addresses can be incremented or decremented or have arithmetic operations performed on them in the accumulator to give other addresses. The Electron, BBC Micro, Vic 20 and Commodore 64 have eight-bit index registers whose contents are added to an address to give a further address.

Whichever way round it is done, this is known as *indexed addressing*. It is most commonly used when reading data byte-by-byte out of a table. The program will start with the address of the first byte of the table, then work its way through it by incrementing the index register each time.

PUTTING UP THE FLAGS

In each of these home computers, one eight-bit register stores what are known as *flags*. These are single bits that are set, or reset, to 1 or Ø depending on the outcome of certain operations.

- WHAT THE CPU IS AND WHAT IT DOES
- COMMUNICATING WITH THE MICROPROCESSOR
- THE REGISTERS
- HOW THE COMPUTER KEEPS TRACK OF A PROGRAM
- WHAT FLAGS INDICATE
- MEMORY LOCATIONS AND THE STACK

Each flag has its own particular meaning. For example, the zero bit, the least significant bit of the register—that is the one at the right-hand end in the normal convention of writing—is the *carry* or C flag. If you do an addition or subtraction—or other logical or arithmetic operations that will require a bit to be borrowed or carried—then the C flag is set to 1.

Other bits indicate whether the result of a certain operation was zero (the *zero* flag), or was negative (the *sign* flag), or overflowed the register (the *overflow* flag). These flags are common to all the machines, though they are not necessarily in the same position in the flag register. And there are other flags that are specific to each machine.

Flags are used in conditional statements—the equivalent of IF . . . THEN in BASIC. In the machine code equivalent, you write instructions which will make the machine jump or branch if the sign or the zero flag was set, say.

THE STACK

Another register all these computers have is the *stack pointer*. It holds the address of a memory location. And like all pointers this address marks the end of an area of memory.

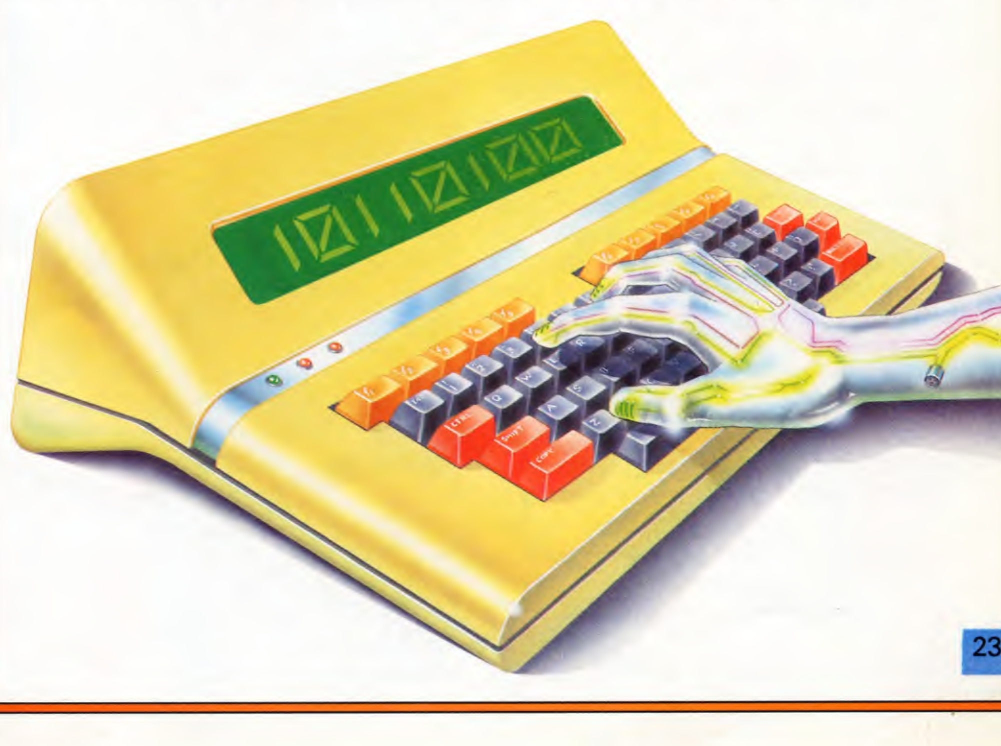
That specialised area of memory is, of course, the *machine stack*. This is a quick-access area where information can be stored temporarily and recalled without the normal addressing procedures.

The stack is like an ordinary pile of papers. When you want to put information into it, it has to be piled on the top. And when you want to take something out of it, you have to take it from the top. The rule is last in, first out.

The microprocessor will only put things onto the top of the stack, and if it is instructed to take something from the stack it will always take the top item. This means that you don't have to specify the memory location you are interested in, or its position in the stack. At any one time there is only one top location, so the microprocessor cannot get confused.

But *you* can. It is up to you to remember which is in each memory location in the stack and to make sure that the data you want is at the top of the stack when you want to use it.

Now that you have grasped the concept of the stack being a pile you can put things on the top of and take things off the top of, there is some bad news. In all the machines under



Microtip

How to count in computerese

When you count things on a computer you start at zero and work upwards, for example, memory location are numbered from ØØØØ to FFFF. The same goes for the bits in a byte. When you number the bits you start at the righthand end and number from nought.

So the extreme righthand binary digit is called the zero bit, or bit zero. The next one to the left is called bit one, the next bit two and so on, up to bit seven—the extreme lefthand bit in a byte.

In some cases, the terms 'righthand' and 'lefthand' can be a little confusing, especially when dealing with two-byte numbers which can be stored either high-low or low-high, depending how your computer operates. In proper 'computerese', these are called the most significant—that is the one with higher value—and the least significant. So if you are storing an address—3D8E, say—3D is the most significant byte and 8E the least significant byte.

The Sinclairs, Commodores and Acorns put the low or least significant byte—8E here—into the lower memory location. The high byte, which is worth 1ØØ hex or 256 decimal times the value of the low byte, 3D in this case, is put into the memory location with the higher address—that is the memory address storing 8E plus 1. There is one exception to this low-high convention—BASIC line numbers are stored high-low.

The Dragon and Tandy store *all* their two-byte numbers with the high byte in the lower memory location and the low byte in the higher memory location.

The same terminology applies to bits. In the binary number Ø1Ø1Ø1Ø1 the most significant bit, bit seven is Ø and the least significant bit, bit zero, is 1.

consideration here, the base of the stack is towards the top of RAM, with the rest of the stack piled, item by item, down underneath it. So, in fact, you put things onto, and take things off, the *bottom* of the stack.

This means that when a new item of data is pushed onto the stack, the stack pointer is decremented, and when a byte is pulled off the stack pointer it is incremented. Otherwise, the pile concept holds true, with the base of the stack fixed by a system variable and items of data added to or subtracted from the free end one at a time.

SUBROUTINES

One of the most common uses of a stack is to keep track of your place in a program when the computer goes off to execute a subroutine. When the computer hits an instruction that tells it to jump to a subroutine, the contents of the program counter are pushed onto the top of the stack.

The address of the beginning of the subroutine is then written into the program counters. As each instruction in the subroutine is executed the program counter is incremented in the usual way—until it hits a return instruction. Then data at the top of the stack is pulled back into the program counter and incremented in the normal way.

This explains why, in BASIC programs, each nested subroutine must be completely contained in the one preceding it, otherwise the wrong departure addresses will be on the top of the stack.

Now that you have some grasp of the more important concepts behind machine code programming you can find out how they relate to the chips in your computer.

There are 21 *user registers* in the Spectrum and ZX81's Z80 chip. They are called user registers because they can be controlled by the user, or programmer, of the machine.

There are two eight-bit accumulators, A and A′. These cannot be used at the same time, but you can switch between them to perform two operations simultaneously. It is not advisable to try this on the ZX81, though. If the machine is in SLOW mode, the picture will be lost.

The flag register, F, is also eight-bit, although not all of the bits need concern you as a programmer. The zero bit is the carry flag C; bit one is the subtract flag N, which is only brought into play when *binary coded decimal*, a special binary representation of decimal, is used; bit two is the parity/overflow flag P/V; bit four is the half-carry flag H which is also used in binary coded decimal; bit six is the zero flag Z; and bit seven is the sign flag S. There is an alternate flag register F′ which is switched in with A′.

There are six general purpose registers, BC, DE and HL. These can either be used separately as eight-bit registers, or in pairs as 16-bit registers. And there is also an alternate set—B′C′, D′E′ and H′L′. The HL and the alternate H′L′ registers can also be used as 16-bit accumulators. The H′L′ register must be restored to its former value before returning to BASIC, though.

The stack pointer is 16 bit, as are the index registers, IX and IY. These hold addresses which can be incremented or offset by adding or subtracting numbers known as *offsets*, to give another address. This facility is often used to read a table of data from a base address.

IY usually points to the centre of the systems variables and is used by ROM routines to index them. If the IY register is used, its original value must be restored afterwards or your computer will not work.

There are two other special purpose registers on the microprocessor, the I and the R. The I register is used to store part of the address used to initiate *interrupt routines*. These are routines which interrupt the normal flow of your machine code program every 50th of a second to perform some vital function like scanning the keyboard.

The R register is the *refresh* register, but you do not use it on the Spectrum or ZX81.

The bottom of the stack is at RAMTOP and it can occupy as much of the RAM as required. The stack pointer, or SP register, is 16-bit as it stores the whole of the address of the last occupied byte of the stack.

Q+A

When would I use binary coded decimal?

BCD is used when you need to print numbers on the screen very quickly. Unlike hex, BCD does not have to be converted into decimals before mere mortals can understand it, it is already in decimal form.

To encode a two-digit decimal number into a binary byte, you put the right-hand digit into the least significant four bits—known as a *nibble*, sometimes printed as nybble—and the left-hand digit into the most significant nibble.

This doesn't use all the possibilities—there are after all 16 possible digits that can be encoded into four binary bits, not just ten. So it is wasteful of space. BCD is really hex without the letters.

The only problem this creates is when you get a number bigger than 9 in one of the nibbles. This is where the half carry flag comes in. (The Acorn computers carry numbers automatically.)

The Electron, BBC Micro and Vic 20 use the same chip, the 6502, while the Commodore 64's 6510 chip is very similar and the way it works is virtually identical.

Both these chips have a single eight-bit accumulator and two eight-bit index registers, X and Y. They contain *offsets* which are added to an address to give another address. This method is often used to read a table of data. The address here would be the base address of the table. To read the table a loop would be constructed which incremented the offset each time the computer went round. This would direct the program to the next byte of the table each time the loop was performed.

It is possible to direct a machine code program to an address which contains another address—though this must be in the zero page, as the contents of any memory location can only be eight bits long. You can offset either of these two addresses, but the X register must be used to offset from the first address and the Y register to offset from the second address.

The eight-bit P or *processor register*, known on the Commodores as the *status register*, contains the flags. The most important are the carry flag C which occupies the zero bit; the zero flag Z in bit one; the overflow flag V in bit six and the sign flag N in bit seven.

There are three other flags whose use you may come across. The decimal flag D, which is bit three, is set when the microprocessor is doing arithmetic in *binary coded decimal*, which is a binary representation of normal decimal figures. The break flag and the interrupt flag, B and I, are used in *interrupt routines*. These are routines which interrupt the normal flow of your machine code programs at regular intervals to perform vital functions like scanning the keyboard to see if a key has been pressed.

The stack is confined to page one, with its base at &Ø1FF, so the stack pointer or SP register only has to be eight bits long. It can contain any number from &ØØ to &FF and gives the least significant byte of the first free address below the bottom of the stack. The microprocessor already knows the most significant bit of the address must be Ø1 as the stack is on page one.

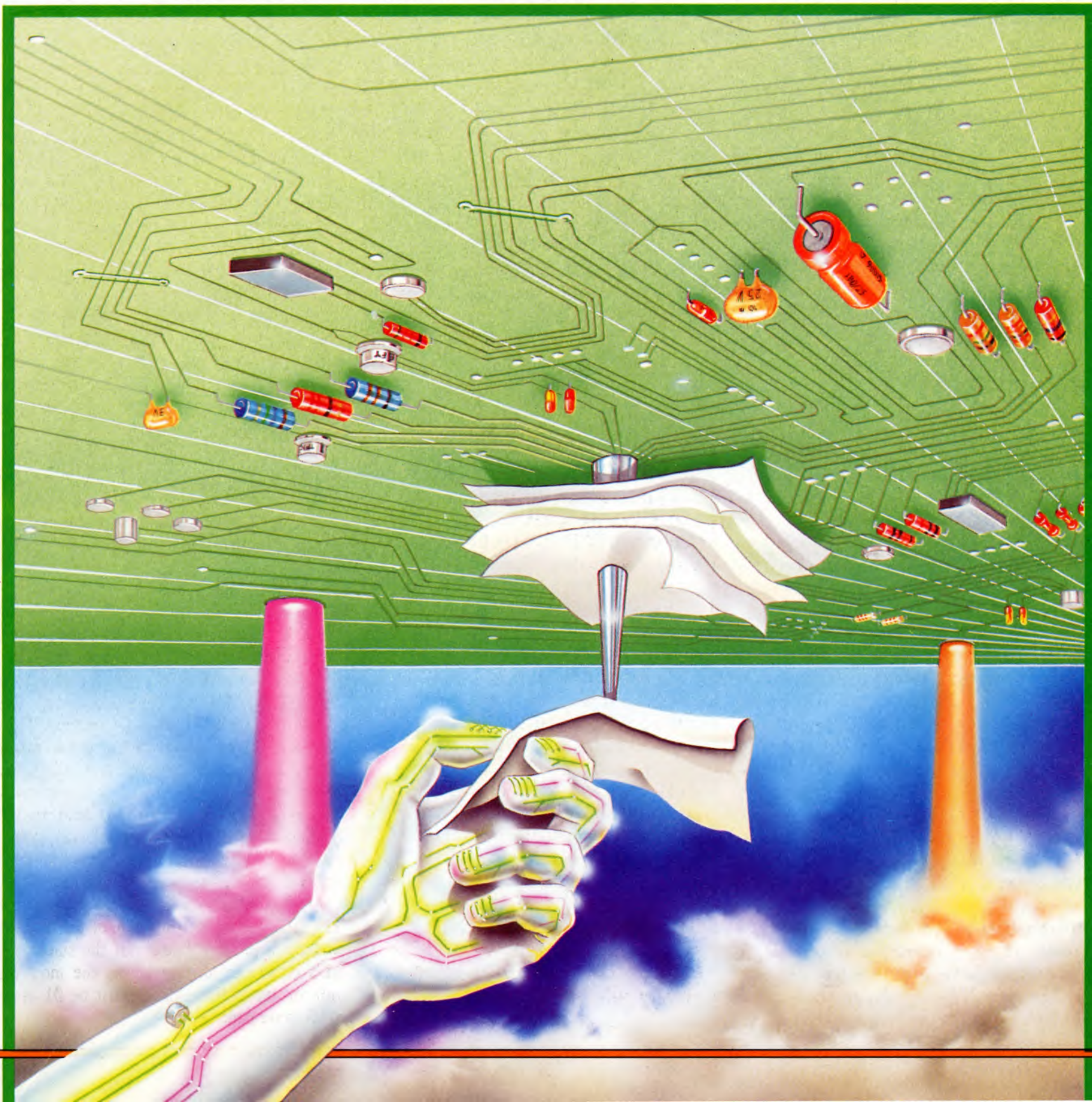
The microprocessor on the Dragon and Tandy has two accumulators, the A register and the B register, which can be used independently as separate eight-bit registers, or together as a 16-bit accumulator D.

It has two 16-bit index registers, X and Y. These hold addresses which can be incremented or offset by adding or subtracting numbers, known as *offsets*, to give another address. This is often used to read through a table of data from a base address.

The *condition code register* holds the flags. The zero bit is the carry flag C; bit one is the overflow flag V; bit two is the zero flag Z; bit three is the sign flag N; bit five is the half-carry flag which is brought into play when *binary coded decimal*, a special binary representation of decimal numbers, is used; and bits four, six and seven—the normal interrupt mask flag I, the first interrupt mask flag F and the entire flag E—are used in *interrupt routines*. These are routines that interrupt the normal flow of your machine code program at regular intervals to perform vital functions like updating the TIMER.

The Dragon and Tandy's 6809 chip also has an eight-bit *DP*—or *direct page—register* which stores the most significant byte of the addresses you choose to be on the direct page so that any location on that page can be addressed by its least significant byte only.

These are two 16-bit stack pointers, S and U. The S pointer points to the last full address on the hardware stack whose base is at RAMTOP, or &H7FFF, unless it has been pushed down to make room for a machine code program. The U pointer is used with the user stack, whose base is usually defined by the user. The user stack is not used very often and the U register is often used as an extra 16-bit index register by the more sophisticated Dragon and Tandy machine code or assembly language programmer.



UNDERSTANDING PEEK AND POKE

Here's your chance to have direct control over your computer by looking straight into its memory and altering or using the values it stores. And you can do it all in BASIC too.

Most of the time you can use a computer without any need to be concerned with how its memory works. When you write LET A = 67 for instance, the computer sorts out for itself suitable free memory locations, labels them 'A' and stores the number 67 in them. Then, when you type PRINT A, the computer knows exactly where to go to find it again. It does all this automatically, and it is only when you come to learn machine code that you *have* to tell the computer which memory locations to use.

But there is a way, in BASIC, to look at the computer's memory, and you can actually use the values stored there in your programs. You can also put in your own values to alter the way the computer behaves.

The BASIC tools to do this are PEEK and POKE. PEEK allows you to look at the value stored in memory, POKE is used to put in your own value. The Acorn computers use the query ? and sometimes the pling ! instead of PEEK and POKE, but the end result is the same.

HOW PEEK AND POKE WORK

The Commodore, Dragon, Tandy, 48K Spectrum and Acorn computers all have 65536 addressable memory locations, or 64K. The 16K computers have 32768 locations. Some of this is ROM, which is the Read Only Memory. The contents of the ROM are fixed so that while you can look at it using PEEK, you cannot POKE anything into it to alter what is already there. The rest of the memory is the RAM—the Random Access Memory, or Read Write Memory as it is sometimes known. You can PEEK and POKE this section of memory, and this is where BASIC programs and variables are stored.

This program allows you to look at any of the computer's memory locations—the whole of the ROM and the RAM:

```
1Ø INPUT "ADDRESS...";A
2Ø LET N = PEEK(A)
3Ø PRINT "CONTENTS..";N
4Ø GOTO 1Ø
```

```
1Ø INPUT "ADDRESS...",A
2Ø N = ?A
3Ø PRINT "CONTENTS..";N
4Ø GOTO 1Ø
```

PEEK 1246Ø



- LOOKING INTO YOUR COMPUTER'S MEMORY
- HOW TO USE **PEEK** AND **POKE**
- **POKE**ING CHARACTERS ONTO THE SCREEN
- READING THE SPECTRUM'S TIMER
- COLOURFUL DRAGON AND TANDY
- RETRIEVING A LOST PROGRAM ON THE ACORN
- CONTROLLING THE COMMODORE

Input any number you like between Ø and 65535 and you'll see what's in that location—although if you try to PEEK into *some* areas of memory you may be given a dummy value rather than the real value stored there.

The contents of the RAM will depend on what you were doing with your computer but the contents of the ROM are fixed.

Notice that the numbers are always integers between Ø and 255. In hex this is from Ø to FF, which means they are all single bytes (a byte is a two-digit hex number). Each memory location will hold one byte, and any larger number cannot be stored in one memory location. If you add 1 to FF in hex you'll get Ø1ØØ. This is two bytes and so needs to be stored in two memory locations: Ø1 in one location and ØØ in another.

The next program lets you POKE numbers into the memory. Only POKE single bytes at this stage, that is any number between Ø and 255.

```
1Ø PRINT "CONTENTS....";PEEK(3ØØØØ)
2Ø INPUT "NUMBER......";N
3Ø POKE 3ØØØØ,N
4Ø PRINT "NEW CONTENTS";PEEK(3ØØØØ)
45 PRINT
5Ø GOTO 2Ø
```

(For the Vic, change location 3ØØØØ to 9ØØ.)

```
1Ø PRINT "CONTENTS....";?3ØØØ
2Ø INPUT "NUMBER......",N
3Ø ?3ØØØ=N
4Ø PRINT "NEW
CONTENTS";?3ØØØ
45 PRINT
5Ø GOTO 2Ø
```



The program first prints out the contents of the location then POKEs your number into it and prints out the contents a second time to prove that your number really is there.

You can change the memory location to any other location you like. Notice that nothing happens if you try to POKE into the ROM, although you won't do any harm if you try. You may crash the system if you POKE into certain areas of RAM but again, it won't do any lasting harm—simply switch off the machine for a moment to reset the memory.

Now try POKEing a number bigger than 255 and see what happens. On the Dragon, Tandy, Commodore and Spectrum you'll get an error message because you can only get one byte into each location. On the Acorn computers, though, things are different. The computer accepts the number but only the last or least significant byte is stored. So if the number was 26Ø—which is Ø1Ø4 in hex—you'll find that only the Ø4 part is stored.

POKEING ONTO THE SCREEN

If you look at the memory maps on pages 208 to 215 you'll find that a section of the memory is devoted entirely to the screen display. If you POKE certain numbers into that area then characters will actually appear on the screen. To see a particular character you have to POKE its ASCII code on the Dragon, Tandy and BBC computers, and its screen code on the Commodore. The Spectrum uses a different method described below. Try these lines:

```
POKE 1Ø24,1: POKE 55296,3
```

```
POKE 768Ø,1:POKE 384ØØ,3
```

```
MODE 7: PRINT: ?&7CØØ=65
```

```
POKE 1Ø24,65
```

You should see the letter A appear at the very top left-hand corner of the screen.

On the BBC and Dragon it is important that the screen has not scrolled before you try POKEing characters or they could appear anywhere. So clear the screen first by typing CLS then RETURN or ENTER. On the BBC a change of mode will also reset the screen. Even typing MODE 7 when you are already in MODE 7 will still work. Note that you cannot POKE a character like this on the Electron as it only works in the teletext mode—MODE 7.

The reason you need two POKEs on the Commodore is that the first puts the A on the screen and the second prints it in a colour you can see. It would be in the background colour otherwise. You may be wondering what advantage there is in POKEing characters on the screen when up to now you have managed quite happily with PRINT AT, PRINT TAB or PRINT @. Using PRINT to position characters is usually the easiest and most convenient method, but there are some cases where it is better to POKE them. For example, if you PRINT anything at the last screen position, then the screen will always scroll—but you can POKE characters here without any problem. This is especially useful in games when you want to move a character over the whole screen.

SPECTRUM CHARACTERS

On the Spectrum, things are a bit different as it is not possible to POKE a whole character on the screen. A character, like a UDG, is made up of dots on an 8 by 8 grid. And each line has to be POKEd separately onto the screen in order to build up the whole character.

Luckily, the shapes of the characters are stored in the ROM, so you can PEEK into the ROM to have a look at each line and then POKE it onto the screen. The next program POKEs an A at the top left of the screen:

```
1Ø LET dest=16384
2Ø FOR a=1588Ø TO 15887
3Ø POKE dest,PEEK a
```



```
40 LET dest = dest + 256
50 NEXT a
```

Line 1Ø sets the screen address for the first line of the characters. The FOR . . . NEXT loop steps through the area of ROM where the character is stored, and Line 3Ø prints that line of the character on the screen. Line 4Ø increments the screen address by 256 which brings it down directly below the first, ready to print the next line.

This method works, but it is rather slow and so it is almost always better to use PRINT AT or PRINT TAB on the Spectrum.

PEEKING THE ROM AND RAM

When you RUN the first program to look at the computer's memory all you can get out of it are numbers between Ø and 255. However, many of these numbers are actually ASCII codes of letters, some of which make up words and even whole sentences.

Look at the box on page 243 if you're not sure what ASCII codes are.

The next program reads through the whole of the computer's ROM and converts the numbers into characters before printing them on the screen:

```
10 FOR A=0 TO 16383
20 LET N=PEEK A
30 IF N>31 AND N<127 THEN
   PRINT CHR$N;
40 NEXT A
```

```
10 FOR A=40960 TO 49159
20 N=PEEK(A)
30 IF N>31 AND N<91 THEN
   PRINT CHR$(N);
40 NEXT A
```

For the Vic, change Line 1Ø to:

```
10 FOR A=49152 TO 57347
```

```
10 FOR A=&8000 TO &FFFF
20 N=?A
30 IF N>31 AND N<127 THEN
   PRINT CHR$(N);
40 NEXT A
```

```
10 FOR A=&H8000 TO &HBFFF
20 N=PEEK(A)
30 IF N>31 AND N<127 THEN
   PRINT CHR$(N);
40 NEXT A
```

Lines 1Ø and 4Ø step through the whole of the ROM, Line 2Ø PEEKs at each location, and Line 3Ø converts the number into a character and prints it out. Line 3Ø also limits the range of numbers that are converted so the computer doesn't try to print out control codes or graphics symbols.

You can also print out the contents of the RAM in exactly the same way—simply by changing the memory addresses in Line 1Ø. The addresses given below print out part of the area where BASIC programs are stored. This lets you see the program itself as it is actually stored in the memory.

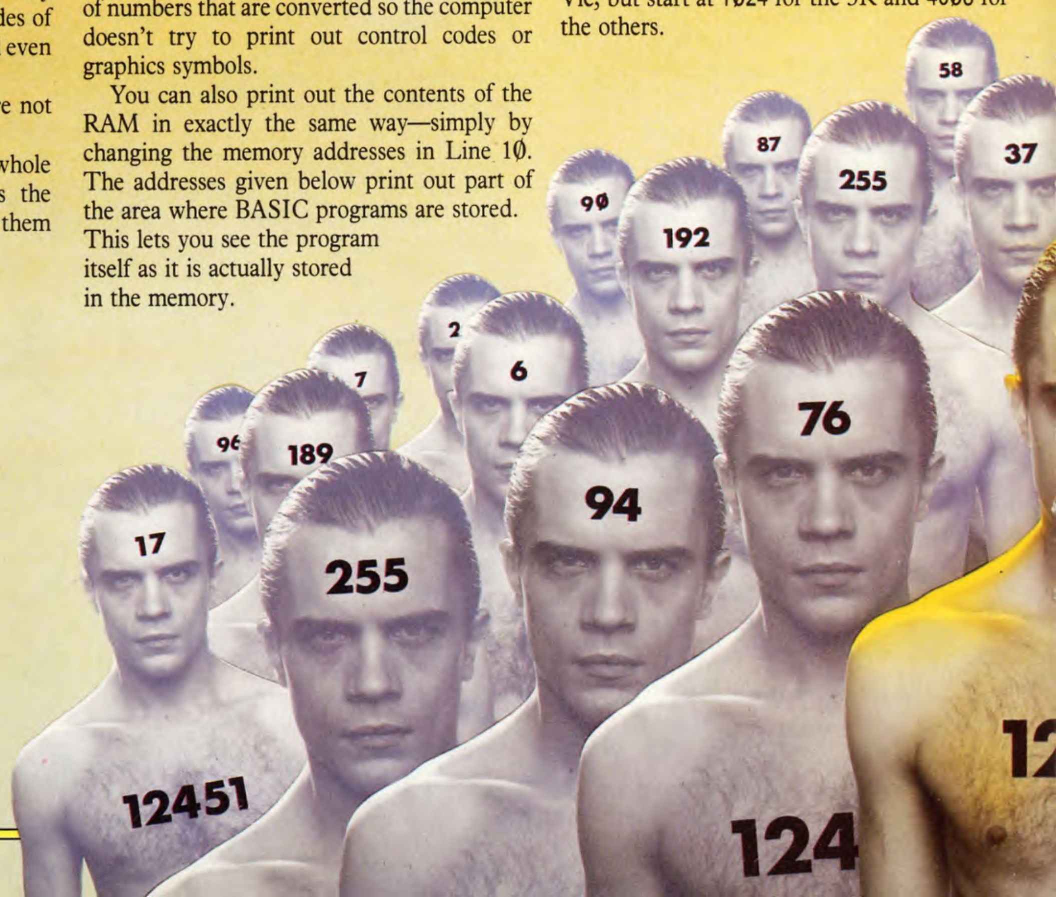
It is best to turn off the computer for a second before trying out this program to make sure that the RAM isn't cluttered up with all sorts of other programs. Here is the new Line 1Ø:

```
10 FOR A=23755 TO 65000
```

```
10 FOR A=16510 TO 30000
```

```
10 FOR A=2048 TO 40959
```

Use FOR A=4Ø96 TO 7679 for the unexpanded Vic, but start at 1Ø24 for the 3K and 46Ø8 for the others.





T

```
10 FOR A=&H1E00 TO &H1F00
```

```
10 FOR A=PAGE TO PAGE+255
```

Try looking at other areas of RAM in the same way.

USING PEEK AND POKE

So far you've been looking at the computer's memory in general and POKEing characters onto the screen. This gives you a good idea of how PEEK and POKE work but it is not particularly useful in itself.

To do something really useful you have to look at specific memory locations. For example, on the Spectrum, memory location 23609 controls the sound that the keyboard makes when you press a key. Normally it contains 0 which makes a short click, but a larger number—you can go up to 255—makes a longer beep. POKE 23609,80 makes a reasonable sort of blip noise.

What you can do with PEEK and POKE depends mainly on the computer you have.

ASCII CODE TABLE

Each character the computer uses has a code number and most computers use a standard code called the ASCII code. ASCII stands for American Standard Code for Information Interchange—pronounced ass-key for short.

These codes are the numbers used with CHR$ and ASC (or CODE on the Spectrum). CHR$ converts the number into a character while ASC or CODE does the reverse and converts a character into its code number.

Also, whenever the computer stores a word in its memory it is the ASCII code of the characters that are stored.

Here's a chart of the ASCII codes:

ASCII chart

Code number	ASCII character	Code number	ASCII character
32	space	62	>
33	!	63	?
34	"	64	@
35	#	65	A
36	$	66	B
37	%	67	C
38	&	68	D
39	'	69	E
40	(	70	F
41	)	71	G
42	★	72	H
43	+	73	I
44	,	74	J
45	–	75	K
46	.	76	L
47	/	77	M
48	0	78	N
49	1	79	O
50	2	80	P
51	3	81	Q
52	4	82	R
53	5	83	S
54	6	84	T
55	7	85	U
56	8	86	V
57	9	87	W
58	:	88	X
59	;	89	Y
60	<	90	Z
61	=		



For example, the other computer keyboards don't make a noise when a key is pressed so there's no memory location that controls the keyboard sound. Also, what takes six or seven PEEKs and POKEs on one computer may be done with a single BASIC keyword on another machine.

The Commodore makes the most use of PEEK and POKE. The Spectrum, Dragon and Tandy use them occasionally, often to alter the way the keyboard works or to alter the screen display. On the Acorn machines you may never use them at all in a BASIC program, although you would if you were using assembly language.

Here, then, are some things to try out on your computer.

You've already seen one POKE to alter the sound of the keyboard. The next one alters the length of time before a key starts to auto-repeat—useful to speed up games relying on key presses to move a character. In this, and each of the following examples, X is a variable which you should set when you type in the Lines.

```
POKE 23561,X
```

X is normally set at 35 whenever the computer is switched on so use a number less than 35 to speed up the delay and a larger number—up to 255—for a longer delay.

It is also possible to alter the auto-repeat itself by using:

```
POKE 23562,X
```

This time X is normally 5. So use X = 1 for fast auto-repeat and X = 255 for very slow.

TIMING

The Spectrum doesn't have a TIME keyword so you have to PEEK into the memory to make use of the computer's internal timer. Time is stored as three bytes in three consecutive memory locations.

```
PRINT (PEEK 23672+256*PEEK 23673
   +65536*PEEK 23674)
```

will tell you how many 50ths of a second your Spectrum has been on since the last NEW. Press NEW to reset the timer or POKE Ø into each of the three locations.

MORE POKES

Here are two more POKEs that are useful when you're writing programs that other people will use. The first one makes sure that all letters typed in will appear as capital letters:

```
POKE 23658,8
```

This is useful when you want all inputs to be consistent. Use POKE 23658,Ø to return to normal.

The second one alters the cursor to any of the keyboard characters or even whole keywords depending which number you POKE in:

```
1Ø FOR X=1 TO 255
2Ø POKE 23617,X
3Ø INPUT a$
4Ø NEXT X
```

Line 3Ø says INPUT a, this is just to make a cursor appear on the screen. As soon as you INPUT something the program displays the next cursor. The most interesting ones are around X = 2ØØ to 23Ø. For example X = 21Ø gives you [CONTINUE] as a cursor, and X = 225 gives you a [?].

One common use of PEEK and POKE is to provide an auto-repeat on the keyboard. The Dragon and Tandy normally have no auto-repeat, so using keys to move objects round in a game would be quite tiring. You'd have to keep pressing the key over and over again very quickly. But you can use PEEKs to see which keys are being pressed, allowing you to leave your finger on the key for as long as you like.

This method has already been used in the Game Programming section. What happens when you press a key is that a special code number is placed in one of 6 memory locations. PEEK then checks these locations to see which key is being pressed. It is very easy to use this in your programs; for example, the next program uses the cursor keys to draw on the screen:

```
1Ø PMODE Ø,1
2Ø PCLS
3Ø SCREEN 1,1
4Ø X=127:Y=95
5Ø IF PEEK(341)=223 THEN Y=Y-2
6Ø IF PEEK(342)=223 THEN Y=Y+2
7Ø IF PEEK(343)=223 THEN X=X-2
8Ø IF PEEK(344)=223 THEN X=X+2
9Ø IF X<Ø OR X>255 THEN X=
   -255*(X<Ø)
1ØØ IF Y<Ø OR Y>191 THEN Y=
   -191*(Y<Ø)
11Ø PSET(X,Y,5)
12Ø GOTO 5Ø
```

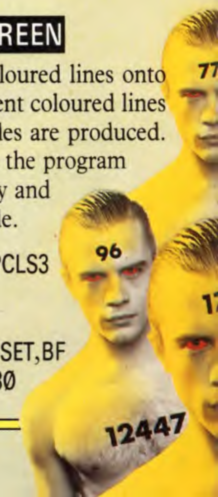
On the Tandy, change the 223 to 247 in Lines 5Ø to 8Ø.

The tables on pages 245 and 246 show what code number is generated by each key and which memory location it is stored in. For example, if A is pressed, then PEEK(339) should equal 251 on the Dragon or 254 on the Tandy, and so on. You can now see where the numbers after the PEEKs for the last program came from.

MULTI-COLOURED SCREEN

The next program POKEs coloured lines onto the screen. By putting different coloured lines close to each other new shades are produced. You can press any key while the program is running to stop the display and see the number for that shade.

```
1Ø PMODE 3,1:SCREEN 1,Ø:PCLS3
2Ø FOR K=Ø TO 255
3Ø POKE 178,K
4Ø LINE(78,46)-(178,146),PSET,BF
5Ø IF INKEY$<>"" THEN 8Ø
```





```
60 NEXT
70 GOTO 70
80 CLS:PRINT K
```

Memory location 178 controls the foreground colour. This normally ranges from Ø to 3, giving four colours, but if you POKE a number higher than K=3 you'll get extra stripey colours. PSET is used in Line 4Ø because we are dealing with foreground colours. For background colours change PSET to PRESET and change the 178 in Line 3Ø to 179.

When you stop the program you can see the value of K for that shade and then you can use that colour in your graphics programs.

SPEED CHANGES

Finally, here's a trick for speeding up the computer—useful if you want to make a game go faster after a player reaches a certain score. Use the first POKE to speed up and the second to return to normal:

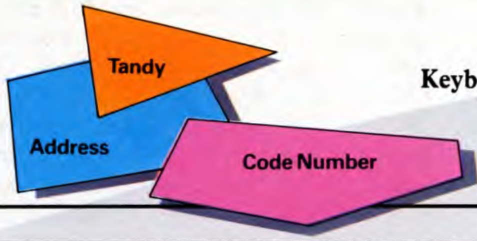
```
POKE 65495,1
POKE 65494,1
```

Keyboard PEEKs and POKEs

Dragon — Address — Code Number

Address	191	223	239	247	251	253	254
338	ENTER	X	P	H	@	8	0
339	CLEAR	Y	Q	I	A	9	1
340		Z	R	J	B	:	2
341		↑	S	K	C	;	3
342		↓	T	L	D	,	4
343		←	U	M	E	–	5
344		→	V	N	F	.	6
345		SPACE	W	O	G	/	7





Keyboard PEEKs and POKEs

	191	223	239	247	251	253	254
338	ENTER	8	Ø	X	P	H	@
339	CLEAR	9	1	Y	Q	I	A
340		:	2	Z	R	J	B
341		;	3	↑	S	K	C
342		,	4	↓	T	L	D
343		–	5	←	U	M	E
344		.	6	→	V	N	F
345		/	7	SPACE	W	O	G

(You can use any number in place of the 1 as they all have the same effect.)

There is only one problem with these POKEs, and that is they may not work on all processors, and may cause the program to crash. But it's worth trying out to see if it works on your machine.

And for those who think that games are too fast, here is a way to slow down output to the screen:

```
POKE 359,6Ø
```

then use POKE 359,57 to return to normal. But don't use this one on the Tandy as it disables the auto-return to the text screen.

PEEKs and POKEs are two keywords which give the Commodore user grass-roots control over the machine. In fact, without them, there's little you can do with the rich selection of sound and graphics features which are so badly supported by the version of BASIC used. And much actual control of the machine can be done in this way too.

A location-by-location memory map (which is provided in the Programmer's Reference Guide) is an essential accessory if you wish to explore the less commonly used POKEs.

At this point, it's worth pointing out that no damage can be caused to the computer itself if you POKE a 'no go' memory location. But you may get lots of meaningless figures if you PEEK a sensitive area. At worst the computer may crash, although even this is a rare event.

Here are some examples which give at least an insight into the sort of control possible through the use of PEEKs and POKEs:

COSMETIC CHANGES

Two of the most frequently used of all POKEs on the Commodore 64 computer are the two which set the screen and border colours:

```
POKE 5328Ø, X for the border
POKE 53281, X for the screen
```

The value X can range from Ø to 15, which corresponds to the colour codes listed in the manual. Try POKEing the value Ø into both locations: the result is a black screen with a blue cursor. You can change this too. Press CTRL and one or other of the colour keys to find a combination you like. White or yellow on black can look much more pleasing than the normal start-up combinations of blues, and are better suited to word processing and datafile handling than more colourful displays.

Other POKEs in this section of memory (53248–54271) relate to the use of the various graphics modes such as sprites because here's where the VIC chip is located.

KEY CONTROL

One of the most useful POKEs for games and drawing programs is the familiar

```
POKE 65Ø,X
```

Where any value of X above 128 switches on an automatic key repeat facility, and where any subsequent POKE of a lower value returns things to normal.

Ever thought how best to protect your programs? Well there's no certain way, of course, but a POKE or two can make life difficult to a casual intruder:

```
POKE 775, 2ØØ
```

This makes it impossible to use LIST in order to inspect a program. Matters can be rectified by POKEing the same location with 167.

Use of the RUN/STOP key can be prevented using:

```
POKE 8Ø8, 239          (POKE 8Ø8, 237)
```

Note that the second POKE listed here cancels the effects of the first—and the remaining POKE on and off forms are shown in this manner too. This second POKE has, of course, got to be entered (in direct program mode) before normal keyboard functions can be resumed. You may find this POKE useful in a program routine that relied on keyboard input. RUN/STOP and RESTORE can also be disabled:

```
POKE 8Ø8, 251          (POKE 8Ø8, 237)
```

Interestingly, the same location is used for both protective methods. Now, this is something that happens quite a few times and suggests an important use of POKEs where multi-function locations are concerned—switching the bits of each byte to high ('on') or low ('off').

The maximum value of 255, for example, is obtained if all bits of a byte are 'high': Let's look at the individual bit values:

bit7	bit6	bit5	bit4	bit3	bit2	bit1	bitØ
128	64	32	16	8	4	2	1

Added together, the values of bits Ø to 7 equal 255. Of the other values used with POKE 8Ø8, the value 237 is obtained only when bit 4 and bit 1 are 'low', or 'off' (in other words, 255 minus 16, minus 2). Value 234 is obtained when bits 4, 2 and 1 are low. This sets the computer back to normal. POKEing 8Ø8 with 239 switches on bit 1, and RUN/STOP is disabled again.

MEMORY MANIPULATION

PEEKs and POKEs are commonly used for memory manipulation so that various areas of RAM can be protected from being overwritten by BASIC. Also, different starting locations can be assigned to, for example, the screen memory locations. And temporary use can be made of locations which normally serve a different functuon (such as the tape in/out buffer). More about this appears on pages 214 to 215 which describes how the Commodore 64's memory is arranged.

POKEable locations can be considered as information storage 'cells' where a particular value can range from Ø through to 255. Of course, the information they contain may act as switches, maybe for screen colour, maybe for a particular sound—but the important distinction is that one location is responsible for only one task. In many cases these have preset, fixed or default values which can be examined using the PEEK program outlined earlier.

Quite often it's useful to clear these locations of any old information or garbage and a simple POKE of a Ø is all that's required.

Another interesting thing happens if you try to POKE a value into an area of ROM/RAM

overlay (such as occurs on the 64 at memory locations decimal 4Ø96Ø and upwards). When you PEEK, you get a value corresponding to BASIC ROM, but when you POKE, you store a figure into the RAM area 'beneath'. This explains why a line such as POKE N, PEEK (N) frequently appears in program listings for the Commodore 64. Here's an example, a program for copying the BASIC ROM into its underlying RAM:

```
1Ø FOR N=4Ø96Ø TO 49151
2Ø POKE N, PEEK (N) : NEXT
```

RUN this and wait for about a minute for the cursor to reappear: BASIC is now in RAM and therefore in a position where it can be changed. But before you can do this you have to flip ROM out of the way. Enter this:

```
POKE 1,54
```

You are actually playing about with perhaps the single most important location in memory—but more on this in later articles. The result is that you are left with a volatile form of BASIC and any POKEs in this area of memory (4Ø96Ø to 49151) can have amusing consequences. In direct mode, enter this string of PEEKs:

```
PRINT PEEK(41229),PEEK(4123Ø),
   PEEK(41231),PEEK(41232)
```

Now press [RETURN] to display four figures: 76, 73, 83 and 212. If you now look up the ASCII table, you will see that these figures correspond to L, I, S and T (the latter being obtained by deducting 128 from 212 to give 84). You've intruded onto the keyword LIST!

Suppose you wanted to change this. The procedure is simply to POKE into the same memory locations the corresponding ASCII codes (with an added 128 for the last one). If, for example, you wanted the keyword SHOW to replace LIST, use this direct command sequence:

```
POKE 41229,83: POKE4123Ø,72:
   POKE 41231,79: POKE 41232,215
```

Press [RETURN] and the job's done. Type SHOW and press [RETURN] to get your former program listing. To return to normal, simultaneously press [RUN/STOP] and [RESTORE] (or use POKE 1,55 if you want to do the same thing within a program). Other keywords can be altered in the same way.

The Acorn computers don't use the keywords PEEK and POKE, they use *indirection operators* instead. These are the query—?, the pling—! and the string operator—$. You've already seen how the query works and the others are explained in the box on page 247. However, the point is, Acorn recommend that you don't use these operators in BASIC programs if at all possible as the program will then not work across the Tube to a second processor. This is because the memory locations will all be quite different when a second processor is fitted.

Virtually everything you want to do in a BASIC program can be done with ordinary keywords. Anything more complicated can be done with the numerous operating system calls.

But there are times, occasionally, when you do need to look straight into the computer's memory, and only the ?, ! or $ will do.

One good example is when you press BREAK and then type ØLD or Ø. by mistake instead of OLD or O. and then try to LIST your program. All you get is Line Ø followed by LD or a dot. The computer thinks you were entering a new program and your old program seems to have vanished—very frustrating if you hadn't yet saved it.

Luckily there is a way to get your program back. When you typed ØLD or Ø. then the first few bytes of your program were corrupted and what was there will be lost for ever, but at least you can retrieve the rest of your program. Type in this Line and all should be well. But don't give it a line number or you'll corrupt even more of your program:

```
FOR A%=7 TO 26Ø: IF ?(PAGE+A%)
   =&ØD THEN !(PAGE+4)=&2Ø2ØF42Ø:
   ?(PAGE+3)=A% ELSE NEXT
```

It works like this. PAGE is the start of the BASIC program, A% is a counter and &ØD is the carriage return code (the computer puts this code at the end of each program line). The program PEEKs at the memory starting at PAGE+7, thus missing out the first seven corrupted bytes. When the carriage return code is found which means the start of a new line, a series of 4 bytes is POKEd into the memory starting at PAGE+4. The four bytes are &2Ø2ØF42Ø which gives two spaces followed by REM (code F4) followed by another space. It starts at PAGE+4 to leave the Ø Line number intact. Finally, ?(PAGE+3)=A% puts the correct length of line marker in the right position.

If you now LIST the program you'll see

```
Ø REM
```

followed by the rest of your program minus the first Line or two.

The last program was useful and practical but here are a couple of fun examples for the BBC computer. They both address the internal control devices, an area you shouldn't really POKE to as you can easily crash the system. But it does no harm to experiment, and if it does crash just press [BREAK] or switch off for a moment.

The first one turns your keyboard into a rather strange musical instrument:

```
?&FE4Ø=Ø
```

The second alters the timers so everything the computer does is slowed down:

```
?&FE45=1 : ?&FE46=Ø
```

In both cases press [BREAK] to reset.

ACORN INDIRECTION OPERATORS

The query (?), is used on the Acorn computers instead of PEEK and POKE, and in the jargon this is known as an *indirection operator*. There are two other indirection operators, the pling (!) and the string ($). The pling gives the contents of four memory locations and allows you to PEEK and POKE four bytes at a time. This is useful when you are storing large numbers. For example,

```
!3ØØØ=99999
```

stores the number as four bytes in four locations starting at 3ØØØ. To PEEK four locations at a time simply use

```
PRINT !3ØØØ
```

The pling is also useful when you're dealing with integer variables—the sort with a percent sign after them, like year%—as these are always stored as four bytes.

The string operator allows strings to be placed in memory. The ASCII code of each character is placed in a single location, with a carriage return code in an extra memory location at the end. Try this:

```
1Ø A=3ØØØ
2Ø $A="WORD"
3Ø PRINT $A: PRINT
4Ø FOR I=Ø TO 4
5Ø PRINT ?(A+I);"□";CHR$?(A+I)
6Ø NEXT I
```

Line 2Ø pokes the string into memory and Line 3Ø PRINTs out all characters starting at location A up to the carriage return code. The next few lines are there to prove that the word really is stored as described. The ASCII code and its character are printed out in turn as the program loops through the memory from location 3ØØØ to 3ØØ4.

By the way, never confuse A$ with $A. The first is simply a string variable named A, while the second is a string operator operating on memory location A.

DRAGON/TANDY: BETTER GRAPHICS



The Dragon and Tandy have five high-resolution graphics modes, or PMODEs, numbered from Ø to 4. They differ in resolution, or the fineness of the screen display, and the colours which you can use. PMODEs may be either two-colour or four-colour, and each has a choice of colour sets.

When you use the four-colour modes—PMODEs 1 and 3—you'll see that colour set Ø consists of Green, Yellow, Blue and Red, whilst colour set 1 consists of Buff (White), Cyan, Magenta and Orange. The SCREEN command allows you to choose the colour set—use SCREEN 1,Ø for colour set Ø and SCREEN 1,1 for colour set 1.

The difference between PMODE 1 and PMODE 3 lies in the resolution they produce. You'll see that graphics produced in PMODE 1 appear coarser than those drawn in PMODE 3 because of the lower resolution. The advantage of PMODE 1 is that it uses less memory space, and the graphics can be a little faster.

When you use PMODE 4 to draw black and white UDGs, every pixel can be switched on and off individually using binary 1s and Øs. In PMODE 3, though, the pixels work in blocks of two, and in PMODE 1 they are in blocks of four. Soon you'll see how 2-bit binary numbers control the colour of these pixel blocks. Table 1 gives a list of the colours and resolutions available in each PMODE.

SETTING UP COLOUR UDGs

Type in this program and RUN it:

```
1Ø PMODE3,1:PCLS:SCREEN1,Ø
2Ø FOR I=1 TO 9
3Ø READ A,B:POKE 18ØØ+I*32,A:POKE
   18Ø1+I*32,B
4Ø NEXT
5Ø FOR K=1Ø TO 22:POKE 18ØØ+K*32,
   192:POKE 18Ø1+K*32,Ø:NEXT
6Ø GOTO 6Ø
7Ø DATA 192,Ø,213,149,213,149,213,
   149,234,17Ø,234,17Ø,213,149,213,
   149,213,149
```

You have already seen how to set up monochrome UDGs (pages 38 to 41). Now add colour to your Dragon or Tandy graphics with useful extra techniques for these machines

- THE DIFFERENT COLOUR SETS AND HOW TO USE THEM
- SETTING UP COLOUR UDGS
- PUTTING UP THE FLAG
- EXPLORING THE PMODES

The program draws a flag and a flagpole on the screen. It works very similarly to the program on page 40 which produced black and white UDGs by taking DATA and POKEing it on the SCREEN. This time, so that you can see the UDG a little more clearly, the DATA has been POKEd into memory locations starting at 18ØØ, which puts it near the middle of the screen, rather than 1536, which would have displayed the UDG at the top left.

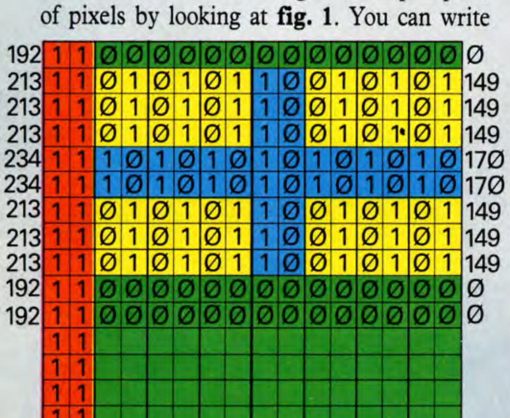
Line 1Ø selects PMODE 3, colour set Ø—Green, Yellow, Blue and Red—and clears the screen ready for the UDG. Lines 2Ø to 4Ø POKE in nine pairs of DATA to draw the flag itself. The rest of the UDG is drawn just by repeating 192 and Ø over and over.

Line 5Ø draws the flagpole by POKEing 192 and Ø 22 times. This is far better than using the original FOR . . . NEXT loop because it saves you having to type in 44 extra pieces of DATA in Line 7Ø.

Line 6Ø is the usual method for keeping the high resolution screen switched on so that the UDG can be seen.

The main difference in the program is in the DATA itself. The DATA is in decimal and was converted from an 8-bit binary number, using the same method for the black and white UDGs on page 38. The difference is that instead of *individual* bits telling the computer to switch *individual* pixels on or off, *pairs* of bits tell the computer which colour to set *pairs* of pixels—in PMODE 1 a pair of bits tells the computer to set two pairs of pixels, one under the other.

You can see how the flag is built up of pairs of pixels by looking at **fig. 1**. You can write two-bit binary numbers on each of the two-pixel blocks—the colours they select will depend on the colour set you have chosen. In colour set Ø you write ØØ on a green block, Ø1 on a yellow block, 1Ø on a blue block and 11 on a red block. Four of these two-bit numbers make up each 8-bit piece of DATA.



If you had chosen colour set 1 by typing SCREEN 1,1 in Line 1Ø, you would have drawn the flag in Buff, Cyan, Magenta and Orange. In this colour set, you write ØØ where you want a buff block, Ø1 on a cyan block, 1Ø on a magenta block, and 11 on an orange block.

Try changing PMODE 3 to PMODE 1 in Line 1Ø. RUN the program and you'll see an elongated version of the flag on the screen. Remember in PMODE 1 each piece of DATA is now controlling four pixels instead of two.

If you were to plan a UDG in PMODE 1 you would work with 2 × 2 pixel blocks and write the 2-bit numbers representing colour on the top line only of each block.

GRAPHICS MODE SIMULATION

Here is a program which will enable you to see what effect decimal DATA has on pixels and pixel colour in each of the PMODEs.

```
1Ø CLS
2Ø INPUT "GRAPHICS MODE (Ø−4)□";MO
3Ø MO=INT(MO):IF MO<Ø OR MO>4
   THEN 2Ø
4Ø INPUT "SCREEN NUMBER (Ø−1)□";ST
5Ø ST=INT(ST):IF ST <Ø OR ST>1
   THEN 4Ø
6Ø INPUT "NUMBER (Ø−255)□";NU
7Ø NU=INT(NU):IF NU<Ø OR NU >255
   THEN 6Ø
8Ø IF MO<4 THEN LE=2 ELSE LE=1
9Ø IF MO<2 THEN DE=2 ELSE DE=1
1ØØ IF (MO AND 1)=1 THEN SP=−2 ELSE
   SP=−1
11Ø FOR K=7 TO Ø STEP SP
12Ø FOR L=1 TO LE
13Ø FOR J=1 TO DE
14Ø IF SP=−1 THEN PP=1358+32*J
   +(3−K)*LE+L:CO=INT(.5
   +(NU AND 2∧K)/2∧K):GOTO 16Ø
15Ø PP=1393+32*J+L−K:CO=INT(.5+
   (NU AND 3*2∧(K−1))/2∧(K−1))
16Ø POKE PP,113−SP*15+CO*(14−SP)
   +ST*64
17Ø NEXT J,L
18Ø POKE PP−32*DE,47+K
19Ø POKE PP−32*DE+1+SP,48+K
2ØØ IF SP=−2 THEN POKE PP+31,112
   −(CO>1)
21Ø POKE PP+32,112+(CO AND 1)
22Ø NEXT K
23Ø PRINT@482,"PRESS ANY KEY TO
   CONTINUE !";
24Ø SCREEN Ø,1−ST
25Ø A$=INKEY$:IF A$="" THEN 25Ø
26Ø GOTO 1Ø
```

When the program is RUN you'll be asked to select a graphics mode and screen number. Next type in a decimal number between Ø and 255. A display showing the colours and relative size of pixel blocks will be shown, with the relevant bit numbers above, and the binary equivalents below.

Use this program to explore the possibilities. In a later article, you'll learn how you can animate your own UDGs.

PMODE NUMBER	PIXELS	COLOUR SET AVAILABLE: SCREEN 1,Ø	COLOUR SET AVAILABLE: SCREEN 1,1
Ø	■■ ■■	Black, Green	Black, Buff
1	■■ ■■	Green, Yellow Blue, Red	Buff, Cyan, Magenta, Orange
2	■■	Black, Green	Black, Buff
3	■■	Green, Yellow, Blue, Red	Buff, Cyan, Magenta, Orange
4	■	Black, Green	Black, Buff

MAKING PICTURES WITH MATHS

Put some of those little-used maths functions into action and see how to use SIN, COS and TAN to find your direction as well as drawing curves, circles and ellipses

Computers are often associated with mathematics, and although BASIC programming requires no great skill in this area many of the processes of BASIC would be very familiar to any mathematician. However, many of them are used not just for maths calculations, but will control all sorts of other operations. Some of the most useful mathematical functions are those which are used to work out the relationship between angles and distances—and these have applications as seemingly far removed from maths as graphics.

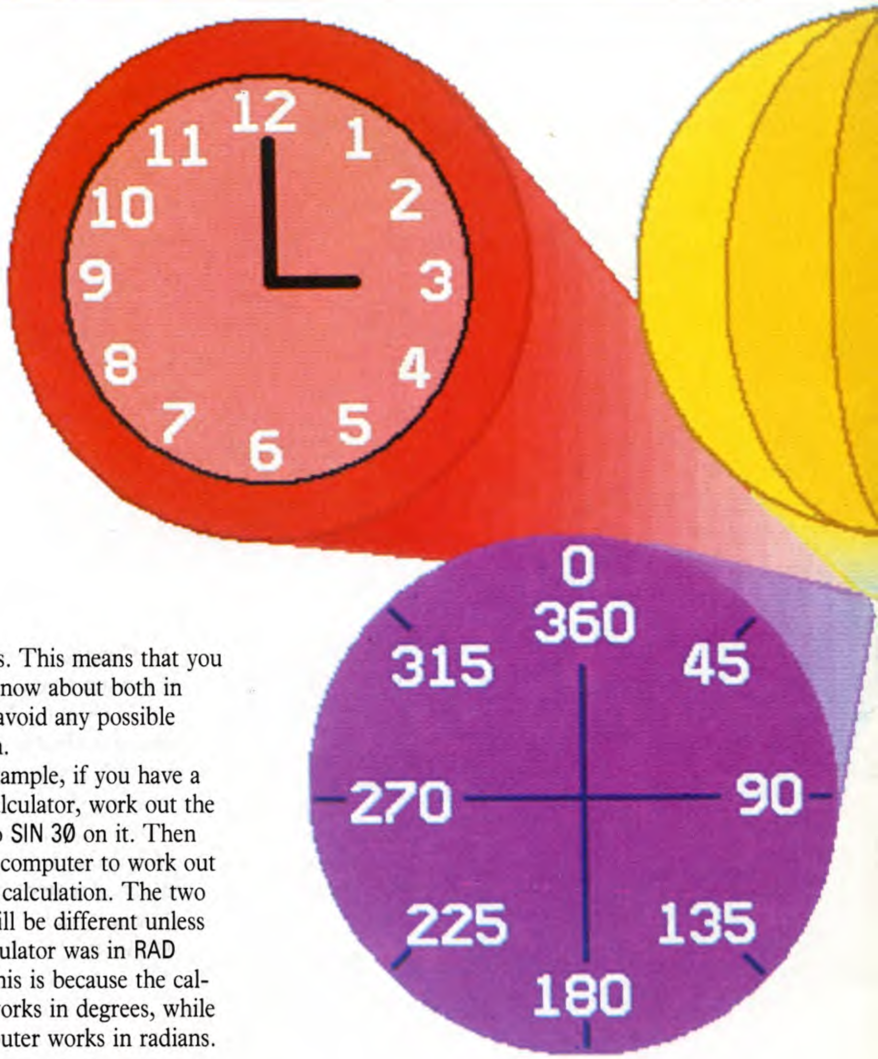
For example, suppose you wanted to draw a clock. You might draw the outline of the clock face using the CIRCLE command, if your computer has one, but you would find it difficult to PRINT the numbers in their correct positions if you had to work out the coordinates of each one tediously by hand. And, of course, a clock without correctly positioned numbers is of little use!

Fortunately, your computer's mathematical functions can be used to take over jobs like this. For example, to return to the clock-face example, you know that the numbers are each positioned one-twelfth of the way round the circle. And your computer can calculate this if you give it a number in degrees or radians—both of which are measurements of the size of an angle.

There are 36Ø degrees in a complete circle and 2 times PI radians in a circle. On the clock face, each number is 3Ø degrees (3Ø°), or PI divided by 6, around the circle from the last—one twelfth of a complete circle, remember. PI (pronounced 'pie') is a number which is often used in calculating various aspects of a circle: such as its area, or circumference (the distance around the circle), for example. PI is roughly 22/7 or about 3.14, and your computer has its value stored in memory (unless you own a Dragon or Tandy, which do not). It is often represented in calculations (and sometimes on the computer keyboard) by the Greek letter π.

You may wonder whether you need to know about both degrees and radians: surely they are both equally useful, and you can make do with using just one? The reason for bothering with both measurements is that although the more common measurement—especially for humans—is degrees, the computers all work in radians. This means that you need to know about both in order to avoid any possible confusion.

For example, if you have a pocket calculator, work out the answer to SIN 3Ø on it. Then use your computer to work out the same calculation. The two results will be different unless your calculator was in RAD mode. This is because the calculator works in degrees, while the computer works in radians.



DEGREES TO RADIANS

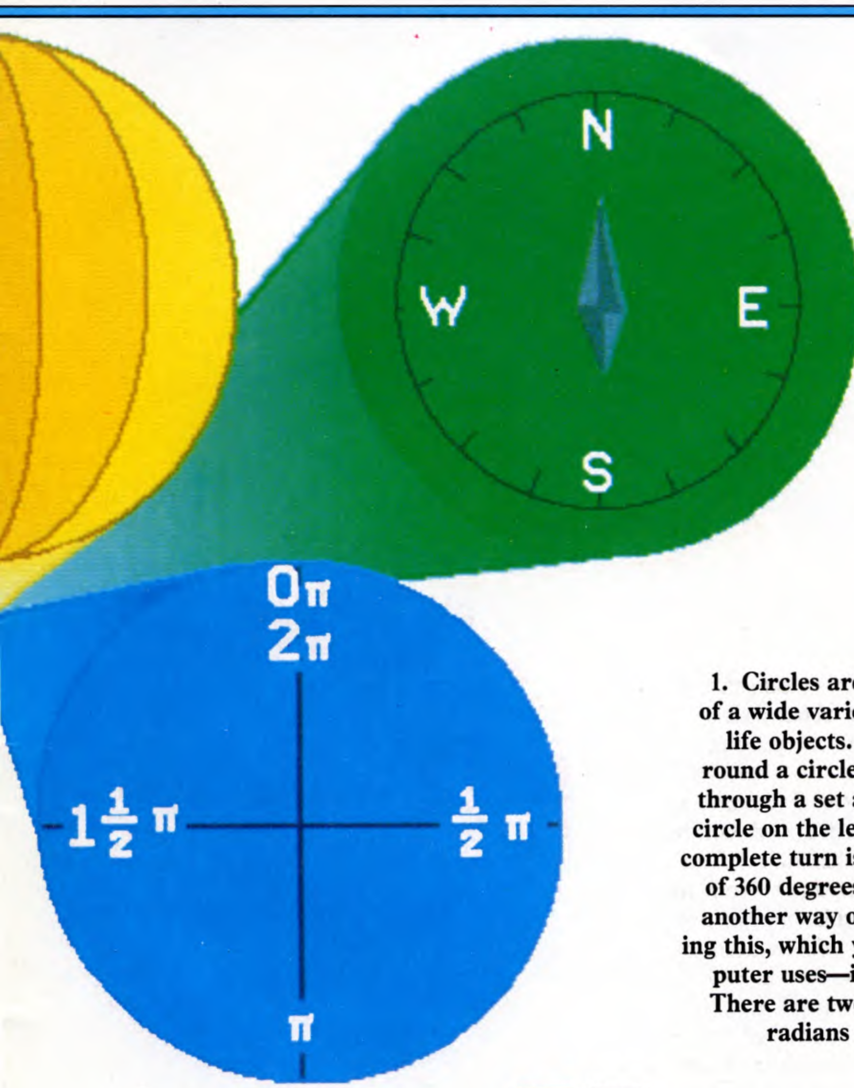
There is a simple way of converting between the two measurements. To change a number in degrees into one in radians, divide the number by 180 and multiply the result by PI. To change a number in radians into one in degrees, do the opposite: multiply by 180, and then divide by PI.

You can get used to these calculations by using the next program. It will convert a number in degrees into one in radians, or vice versa. Try doing calculations on your calculator, if you have one, using the functions SIN, COS, and TAN. Once you have the answer, convert the angle into radians, repeat the calculations, and check your new answer against the computer's.

There is no program for the Acorn computers, as they have two functions (RAD and DEG), which perform the same task.

```
10 INPUT "DO YOU WANT TO CONVERT
   DEGREES INTO RADIANS(1) OR RADIANS
   INTO DEGREES(2)?";a
20 IF a=2 THEN GOTO 80
```

- CONVERTING DEGREES TO RADIANS
- DRAWING A COMPASS
- MEASURING ANGLES ON THE COMPASS
- THE MEANING OF **SIN, COS** AND **TAN**
- DRAWING **SIN** AND **COS** GRAPHS
- USING **SIN** AND **COS** TO PLOT CIRCLES
- BUILDING UP A SPHERE FROM ELLIPSES



1. Circles are the basis of a wide variety of real life objects. As you go round a circle, you turn through a set angle—the circle on the left shows a complete turn is made up of 360 degrees. There is another way of expressing this, which your computer uses—in radians. There are two times PI radians in a circle

```
30 IF a< >1 THEN GOTO 10
40 INPUT "WHAT IS YOUR NUMBER?";b
50 PRINT "THAT IS□";b/180*PI;
   "□RADIANS"
60 PRINT "PRESS ANY KEY TO GO AGAIN":
   PAUSE 0: CLS : GOTO 10
70 INPUT "WHAT IS YOUR NUMBER?";b
80 PRINT "THAT IS□";b*180/PI;
   "□DEGREES"
90 PRINT "PRESS ANY KEY TO GO AGAIN":
   PAUSE 0: CLS : GOTO 10
```

```
10 PRINT "  DO YOU WANT TO
   CONVERT DEGREES INTO"
20 PRINT "RADIANS(1) OR RADIANS
   INTO DEGREES(2)":INPUT A
30 IF A<1 OR A>2 THEN 10
40 INPUT "  WHAT IS YOUR NUMBER";
   B:PRINT "   THAT IS";
50 IF A=1 THEN PRINT B/180*π;
   "RADIANS"
60 IF A=2 THEN PRINT B*180/π;
   "DEGREES"
70 PRINT "   PRESS ANY KEY TO GO
   AGAIN"
80 POKE 198,0:WAIT 198,1:RUN
```

```
10 PI=4*ATN(1)
20 CLS
30 INPUT"DO YOU WANT TO CONVERT
   DEGREES INTO RADIANS(1) OR RADIANS
   INTO DEGREES(2) ";A
40 IF A=2 THEN GOTO 90
50 IF A< >1 THEN GOTO 20
60 INPUT"WHAT IS YOUR NUMBER□";B
70 PRINT"THAT IS□";B*PI/180;
   "□RADIANS"
80 GOTO 110
90 INPUT "WHAT IS YOUR NUMBER□";B
100 PRINT "THAT IS□";B*180/PI;
   "□DEGREES"
110 PRINT "PRESS ANY KEY TO GO AGAIN"
120 A$=INKEY$:IFA$="" THEN GOTO 120
130 GOTO 20
```

When you RUN the program, the computer will ask you first to INPUT a 1 or a 2, depending on whether you want to convert degrees to radians, or vice-versa. It will then ask you for the number you want to convert, and when you INPUT this, will display the answer for you.

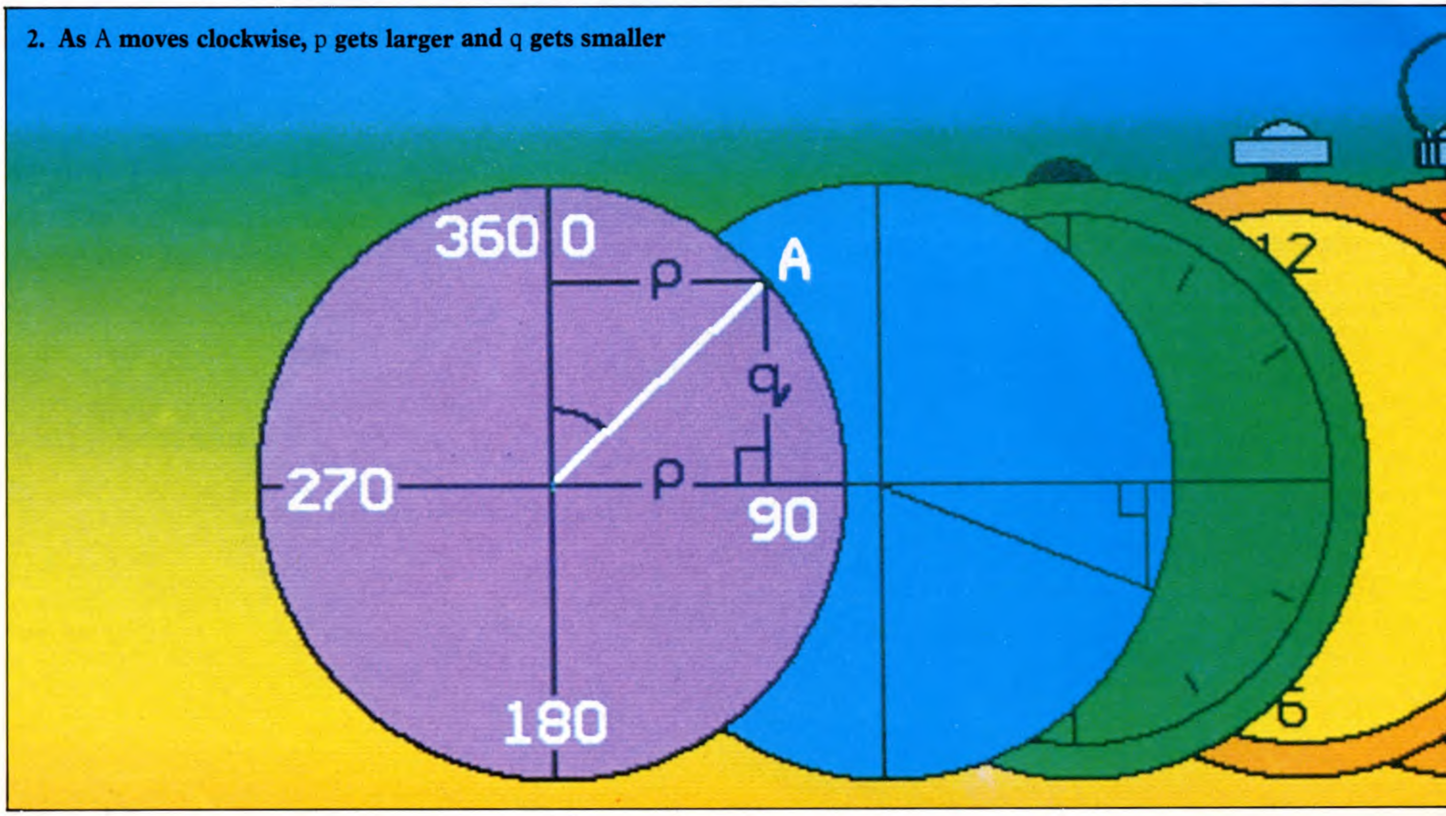
This is all very well, but it can be very difficult to visualize an angle that is only represented by a number—and it is much easier to understand if you can actually see the angle drawn out. One familiar representation of all the possible angles is on the face of a compass, where the numbers round the circumference tell you the angle of the pointer. You can thus check the angle by looking at a real compass, but why bother when you can get the computer to do it for you?

DRAWING A COMPASS

Because there is such a strong connection between a circle and the measurements of an angle you can use such measurements for drawing anything like a clock or a compass which are based on a circle.

The programs below for each computer draw a compass face and place the numbers marking the degrees in their correct positions. North is at 0, East at 90, South at 180, and West at 270 degrees. They will then display any angle you want to see.

The Dragon and Tandy version of this



2. **As** A **moves clockwise,** p **gets larger and** q **gets smaller**

program does not PRINT any numbers around the compass-face. As you know, the computer can't PRINT on the screen in PMODE 4, which is used for drawing the compass. Later in this article, you will see how to overcome this problem.

When you RUN the program, the computer will ask you to INPUT a number. It will draw a line from the centre of the circle to a point, forming an angle of the size you have just told it. So if you INPUT 9Ø, the computer will draw a line from the centre of the circle to the right. If you INPUT 18Ø, it will draw a line straight down from the centre, representing an angle of 180 degrees.

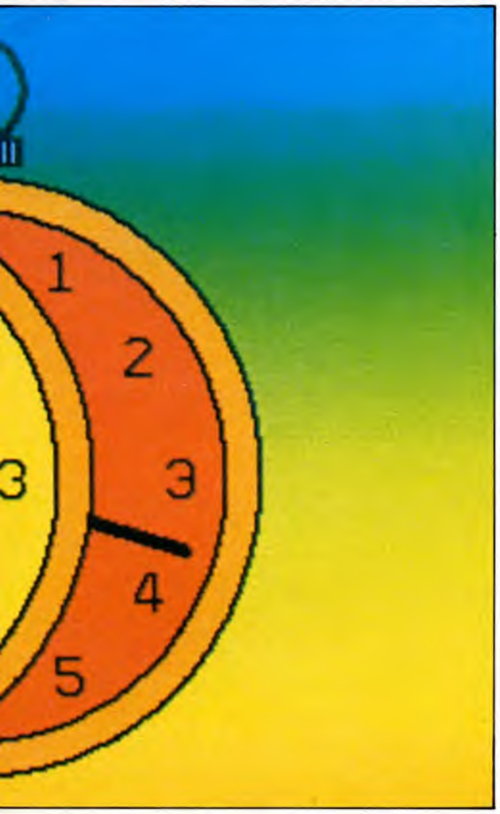
Note that the computer will expect your number to be in degrees, and so will translate it into radians. If you look at each program (except the Acorn version) you will see that the INPUTed variable is divided by 18Ø, and multiplied by PI. This saves you having to make the conversion yourself.

```
1Ø BORDER 4: PAPER 4:INK Ø: CLS
2Ø CIRCLE 131,88,6Ø
3Ø PLOT 131,84: DRAW Ø,8: PLOT 127,
   88: DRAW 8,Ø
4Ø FOR a=Ø TO 2*PI STEP PI/4
5Ø PLOT 131+55*SIN a,88+55*COS a:
   DRAW 1Ø*SIN a,1Ø*COS a
6Ø NEXT a
7Ø PRINT AT 2,16;Ø
8Ø FOR b=45 TO 36Ø STEP 45
9Ø PRINT AT 1Ø-1Ø*COS (b/18Ø*PI),
   15+1Ø*SIN (b/18Ø*PI);b
1ØØ NEXT b
11Ø INPUT "□□□□□WHAT ANGLE IN
   DEGREES□□□□□□□□□□□□
   DO YOU WANT DISPLAYED?",c
12Ø INK 2
13Ø PLOT 131,88: DRAW 45*SIN
   (c/18Ø*PI),45*COS (c/18Ø*PI)
14Ø INK Ø
15Ø INPUT "□DO YOU WANT TO GO
   AGAIN Y/N?";d$
16Ø IF d$="y" THEN PLOT 131,88:
   DRAW OVER 1;45*SIN (c/18Ø*PI),
   45*COS (c/18Ø*PI)
17Ø IF d$<>"y" THEN BORDER 7:
   PAPER 7: CLS : STOP
18Ø PLOT 131, 84: DRAW Ø,8: PLOT
   127,88: DRAW 8,Ø: GOTO 11Ø
```

A standard Commodore BASIC version of this program would be very complex, and so if you have a Simons' BASIC cartridge you should plug it in—the Commodore program is in Simons' BASIC. This applies to the remainder of programs in the article, too:

```
5 INPUT "♥■ENTER ANGLE";A:
   A=A/18Ø*π
1Ø HIRES 1,Ø
2Ø CIRCLE 16Ø,1ØØ,7Ø,7Ø,1
25 TEXT 16Ø,2Ø,"Ø",1,1,1Ø
28 TEXT 157,97,"+",1,1,1Ø
3Ø FOR X=45 TO 36Ø STEP 45
35 TEXT 14Ø+9Ø*SIN(X/18Ø*π),
   1ØØ-9Ø*COS(X/18Ø*π),STR$(X),
   1,1,1Ø
4Ø NEXT X
6Ø FOR X=Ø TO 1.75*π STEP π/4
65 LINE 16Ø+7Ø*SIN(X),1ØØ-
   7Ø*COS(X),16Ø+6Ø*SIN(X),
   1ØØ-6Ø*COS(X),1
7Ø NEXT X
8Ø LINE 16Ø,1ØØ,45*SIN(A)+
   16Ø,1ØØ-45*COS(A),1
1ØØ PAUSE 5:NRM:RUN
```

```
1Ø MODE1
2Ø MOVE 68Ø,912
3Ø FOR A=Ø TO 2*PI+.Ø5 STEP .Ø5
4Ø DRAW68Ø-4ØØ*COS(A+PI/2),512+
   4ØØ*SIN(A+PI/2)
5Ø NEXT
6Ø VDU5
7Ø FOR A=PI/4 TO 2*PI STEP PI/4
8Ø MOVE 63Ø-45Ø*COS(A+PI/2),512+
   45Ø*SIN(A+PI/2)
9Ø PRINT;INT(DEG(A)+.5)
1ØØ MOVE 68Ø-35Ø*COS(A+PI/2),512+
   35Ø*SIN(A+PI/2):DRAW 68Ø-
```




```
   4ØØ*COS(A+PI/2),512+4ØØ*SIN
   (A+PI/2)
11Ø NEXT
12Ø VDU4
13Ø A2=Ø
14Ø MOVE66Ø,512:DRAW7ØØ,512:
   MOVE68Ø,492:DRAW68Ø,532
15Ø VDU3Ø:PRINT' ' "ANGLE"STRING$
   (7,"□"):INPUT TAB(5,2);A
16Ø GCOLØ,Ø
17Ø MOVE 68Ø,512
18Ø DRAW68Ø-35Ø*COS(A2+PI/2),512
   +35Ø*SIN(A2+PI/2)
19Ø A=RAD(A)
2ØØ MOVE68Ø,512
21Ø GCOLØ,3
22Ø DRAW68Ø-35Ø*COS(A+PI/2),512
   +35Ø*SIN(A+PI/2)
23Ø A2=A
24Ø GOTO 14Ø
```

```
1Ø PMODE4,1
2Ø PCLS
3Ø PI=4*ATN(1)
4Ø CIRCLE(127,95),8Ø,5
5Ø FOR X=Ø TO 2*PI STEP PI/4
6Ø LINE(127+72*SIN(X),95-72*COS
   (X))-(127+79*SIN(X),95-79*
   COS(X)),PSET
7Ø NEXT X
8Ø CLS:INPUT" WHAT ANGLE DO YOU
   WANT□";Z
9Ø SCREEN 1,1
1ØØ X=127+6Ø*SIN(Z*PI/18Ø):Y=95
   -6Ø*COS(Z*PI/18Ø)
11Ø LINE(127,91)-(127,99),PSET
12Ø LINE(123,95)-(131,95),PSET
13Ø LINE(127,95)-(X,Y),PSET
14Ø IFINKEY$="" THEN 14Ø
15Ø LINE(127,95)-(X,Y),PRESET
16Ø GOTO8Ø
```

Each program except the Spectrum's begins by setting the correct graphics mode, in Line 1Ø, and then clears the screen. Note that the Commodore requires you to tell the computer what angle you want displayed *before* it draws the compass face, because you cannot print any information on the screen while the computer is in graphics mode.

As the Dragon and Tandy do not have PI stored in their memory, Line 3Ø of the program creates a variable, PI, equal to the value of PI.

Then the programs all draw a circle for the face of the compass. This is done using the CIRCLE command on the Commodore, Dragon, Tandy and Spectrum (Line 2Ø on the Commodore and Spectrum, and Line 4Ø on the others). The Acorn machines have no CIRCLE command, and so the circle is drawn out stage by stage in Lines 2Ø to 5Ø. This routine is explained later.

The programs then print the markings that complete the compass face: numbers marking the degrees in the correct places and short lines at each degree-marking around the circle to show exactly where the marked positions are. All of these are positioned by reference to their angular position, by methods described in detail later on.

The next part of the programs—except on the Commodore—asks you to INPUT your chosen angle: Line 13Ø for the Acorn programs, Line 8Ø in the Dragon and Tandy, and Line 11Ø in the Spectrum version. Because the Commodore INPUT has to be in Line 5, at the beginning of the program, the Commodore program redraws the compass every time you want a new angle.

You will also notice that there is a cross in the centre of the screen. This should help you to check that the angles drawn by the computer are correct: if you ask for an angle of 9Ø degrees, then the right hand horizontal line of the cross should be obscured by the new line. On the Spectrum, Acorn, Dragon and Tandy the cross is drawn using two lines, while the Commodore PRINTs a plus sign in the centre of the screen.

When the computer has displayed the angle you asked to see, it asks you whether you want to check another angle. To mark in your latest choice of angle, it first erases the old line (on the Acorn and Spectrum). The Dragon and Tandy erase the old line after a short pause, generated by a FOR ... NEXT loop. The Commodore does not need to erase the line specifically, since it clears the whole screen when it asks you for your next angle.

The Dragon, Tandy and Commodore all pause after drawing the angle on the compass, before asking you to INPUT another angle. The Acorn program does not pause, but asks you immediately for another angle, but does not erase the last line until you have asked for another. The Spectrum version has a routine asking you whether you want another go—in which case it erases the last line and starts again—or not, in which case it stops.

You could extend the compass analogy by replacing the degree markings with N, S, E, and W, so that you could use the program as a direction finder. For example, suppose you want to travel on a bearing of 27Ø degrees, if you INPUT 27Ø as the angle you want, then the computer will draw a line showing the direction you should take. If you want to experiment, see if you can work out which Lines govern what, and where, is PRINTed around the circle.

POINTS ON A CIRCLE

The programs above all use SIN and COS—two BASIC functions which stand for 'Sine' and 'Cosine'. These are two of the computer's 'trigonometrical' functions. You can forget the long name, since the functions are quite simple.

They refer to the position of a point on the circumference of a circle, in relation to two *axes*—basically, this means how far to right or left it is, and how far up or down it is. The two lines which form a cross in the centre of the circle are the axes. The vertical line is called the 'y' axis, and the horizontal line is called the 'x' axis.

If you look at point **A** on the diagram (**fig 2**) you will see that there is a line connecting it to each axis: these lines are labelled **p** and **q**.

You can see that as the line is drawn at the moment, **p** and **q** are the same length. If **A** moved down to the right along the circumference of the circle, **p** would get larger and **q** would get smaller. And when **A** is at the point marked 9Ø, **q** would be Ø while **p** would be equal to the radius of the circle.

If you use the compass program you can see this. Give the computer angles of Ø, 3Ø, 45, and 9Ø. As the line changes position you can imagine how p and q, although not drawn on the screen, change in length. And if you take a ruler and measure the screen, you can work out what lengths these are.

THE VALUE OF SIN AND COS

For any given angle the ratio between **p** and **q** will always be the same. You can always work out what this ratio is, but it changes as the angle gets larger or smaller.

There is also a ratio between the radius of the circle and **p** and **q**. Remember in the example above, how when **a** was at 9Ø, **p** was equal to the radius of the circle. Like the previous ratio, this will be the same for any given angle, will change as the angle changes, and can always be worked out.

The ratio between the radius and **p** is called the 'sine' of the angle. The ratio between the radius and **q** is called the 'cosine' of the angle. You divide the line (whether it is **p** or **q**) by the radius to get the sine or cosine, so whenever the radius is one unit long, the value of the sine or cosine is *equal* to the length of **p** or **q**.

The triangle below (**fig 3**) is taken from the compass diagram. The wedge shape is formed by the angle between the x axis and point **A** at the centre of the circle. A third line is dropped from point **A** at right angles to the x axis to form the other side of the triangle based on **A**.

On this triangle the sine is the ratio between the *opposite* side of the triangle (the side 'opposite' the angle), and the *hypotenuse*. The hypotenuse is always the side opposite the right angle (see **fig 3**). The third side is known as the adjacent side.

In the same way as the sine, the cosine is the ratio between another pair of the sides. There is a remaining pair of sides which also have a ratio, and this is called the tangent of the angle. The ratios for each are:

sine = opposite/hypotenuse
cosine = adjacent/hypotenuse
tangent = opposite/adjacent

All three ratios can be worked out by your computer, with the functions SIN, COS, and TAN. So, for example, PRINT SIN.5 would print the sine of an angle of .5 radians on the screen.

To refer back to the circle diagram, as the point moves further around the circle in a clockwise direction from the top of the circle, the values of the sine and cosine become larger and smaller respectively, because p and q are changing size. After 90 degrees, though, on the circle, the sine will start to decrease again. And when the point passes 180, things change once more.

Remember the sine measures how far *to the right* of the 'y' axis a point is. This means that in the *left hand half* of the circle, where the point is to the left of the 'y' axis, the sine is *negative*.

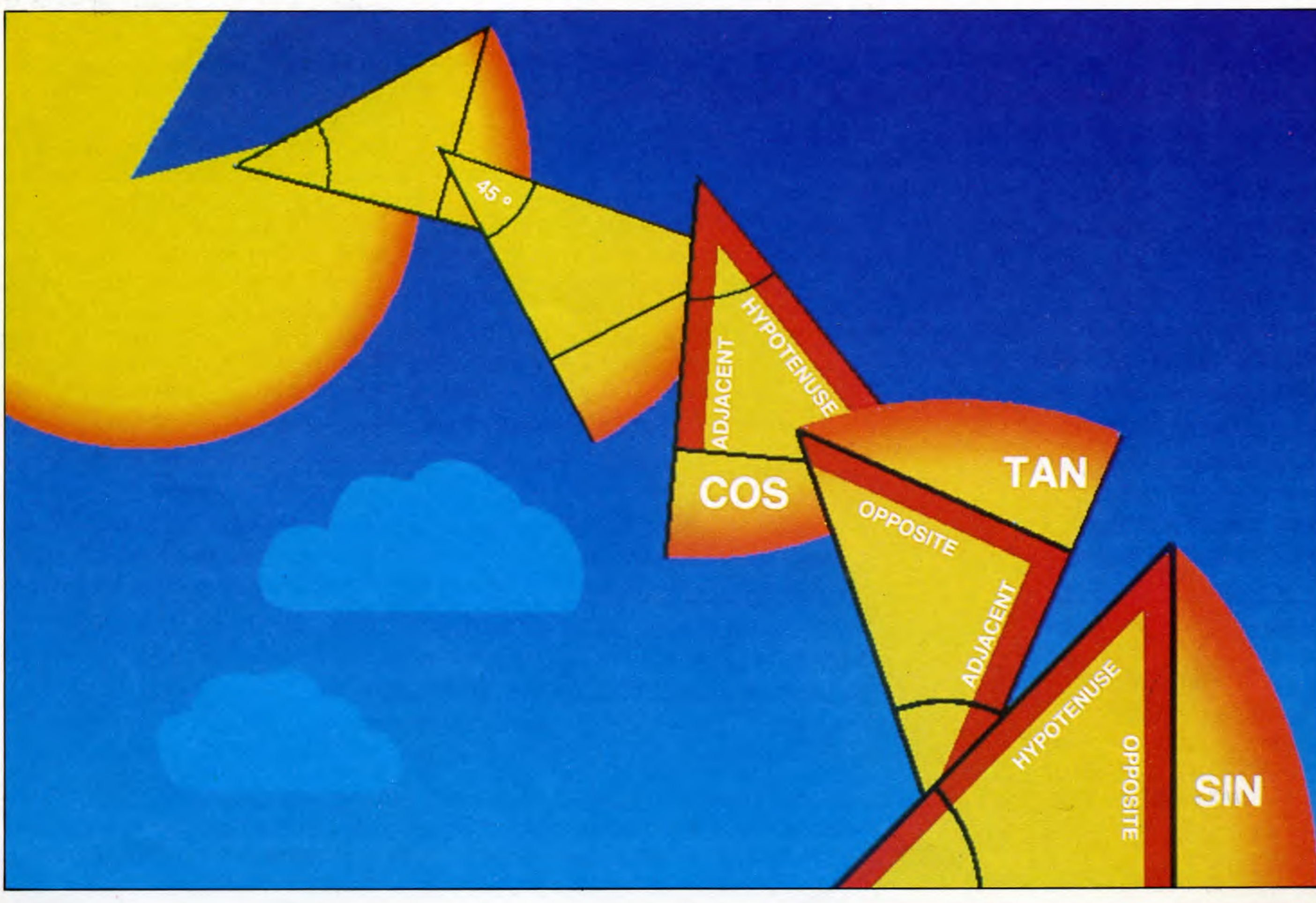
Similarly, as the point goes into the lower half of the circle, where it is below the 'x' axis, the cosine becomes negative. This is because the cosine measures how far *above* the 'x' axis any point is.

DRAWING SIN AND COS GRAPHS

The way the sine and cosine change as the point goes around a number of points on the circle can be made much clearer by plotting a graph of their values for a range of angles. The programs below for each computer will do just that:

```
1Ø PLOT Ø,88
2Ø DRAW 255,Ø
3Ø PLOT 2Ø,Ø: DRAW Ø,175
```

3. The ratio between the lengths of the sides of any right-angled triangle is fixed by the other angles. Depending which sides you compare, the ratio is called SIN, COS **or** TAN





4. A picture of a compass on the Commodore



5. The Spectrum compass showing an angle of 70 degrees

```
40 PLOT 1Ø,158: DRAW 15,Ø: PLOT
   1Ø,18: DRAW 15,Ø
50 PRINT AT 2,Ø;1;AT 2Ø,Ø;−1;AT
   11,Ø;Ø;AT 2Ø,15;18Ø;AT 2Ø,29;36Ø
6Ø FOR a=Ø TO 2*PI STEP .Ø6
7Ø PLOT 2Ø+a*35,88+7Ø*SIN a
8Ø PLOT INK 2;2Ø+a*35,88+7Ø*COS a
9Ø NEXT a
```

```
1Ø HIRES Ø,1:MULTI 3,7,1:COLOUR 6,6:
   LINE 15,1ØØ,36Ø,1ØØ,3
15 LINE2Ø,Ø,2Ø,2ØØ,3:TEXT 1,5,"+1",
   1,3,6
18 TEXT 1,17Ø,"−1",2,3,6:TEXT 6,9Ø,
   "Ø",3,3,6
2Ø FOR Z=Ø TO 2*π STEP.Ø5
3Ø PLOT Z*22+2Ø,1ØØ−SIN(Z)*8Ø,1
35 PLOT Z*22+2Ø,1ØØ−COS(Z)*8Ø,2
4Ø NEXT Z
5Ø GOTO 5Ø
```

```
1Ø MODE1
2Ø DRAW Ø,1Ø24
3Ø MOVEØ,512:DRAW 128Ø,512
4Ø VDU5:MOVEØ,744:PRINT"1"
5Ø MOVEØ,31Ø:PRINT"−1"
6Ø MOVE Ø,5ØØ
7Ø FOR T=Ø TO 36Ø STEP 9Ø
8Ø PRINT;T;
9Ø PLOT Ø,18Ø,Ø
1ØØ NEXT
11Ø VDU4
12Ø X=Ø
13Ø MOVE Ø,512
14Ø FOR T=Ø TO 2*PI+.1 STEP .1
15Ø DRAW X,512+2ØØ*SIN(T)
16Ø X=X+17
17Ø NEXT
18Ø MOVE Ø,712
19Ø X=Ø
2ØØ GCOLØ,1
21Ø FOR T=Ø TO 2*PI+.1 STEP .1
22Ø DRAW X,512+2ØØ*COS(T)
23Ø X=X+17
24Ø NEXT
```

```
1Ø PMODE3,1
2Ø PCLS
3Ø PI=4*ATN(1)
5Ø LINE(6,95)−(255,95),PSET
6Ø LINE(1Ø,Ø)−(1Ø,191),PSET
7Ø LINE(6,45)−(1Ø,45),PSET
8Ø LINE(6,145)−(1Ø,145),PSET
9Ø SCREEN1,1
1ØØ FOR X=72 TO 255 STEP 61
11Ø LINE(X,92)−(X,95),PSET
12Ø NEXT
13Ø FOR X=Ø TO 2*PI STEP PI/123
14Ø PSET(123*X/PI+1Ø,95−5Ø*SIN(X),3)
15Ø PSET(123*X/PI+1Ø,95−5Ø*COS(X),2)
16Ø NEXT
17Ø GOTO 17Ø
```

As with the compass program, the Acorn, Commodore, Dragon and Tandy versions all start off by setting the correct graphics mode: in Line 1Ø. Again as before, the Dragon and Tandy then set a variable for PI using the calculation in Line 3Ø.

Then the computers all draw the axes for the graph, and all except the Dragon and Tandy label each axis. The labels for the y axis (the vertical one) are 1,Ø and −1, which covers all the possible values. The labels for the x axis are degrees from 0 to 360.

The Dragon and Tandy cannot PRINT on a high resolution graphics screen, and so plots short lines at intervals along the axes instead of numbers as labels.

Your computer now PLOTs the graphs, one for the sine, and one for the cosine. It does this using a FOR ... NEXT loop to determine the next angle, before it works out the SIN and COS.

You saw above that there are 2*PI radians in a circle. So, to take in every angle in a circle, the loop is FOR T=Ø TO 2*PI. There is a STEP in the loop so that the computer will not PLOT every tiny angle, but one in three or four, covering the same total angle: 2*PI radians, or 360 degrees.

This STEP makes the program much faster on all but the Acorn computers although the line is dotted as a result. Try removing the STEP to see what difference it makes.

The crucial lines which work out the values of SIN and COS (Acorn 15Ø, 22Ø; Commodore 3Ø, 35; Dragon and Tandy 14Ø, 15Ø; and Spectrum 7Ø, 8Ø) look very complicated, but are in fact quite simple. Each line uses the PLOT, PSET or DRAW command to draw the graph: the Acorn draws a solid line, while the other computers plot a series of points.

The positions at which the computer DRAWs, or PLOTs a point, are determined by the SIN or COS that appears in these lines. The rather complicated looking set of other numbers simply alter the result of the SIN or COS function to scale it to fit the screen position wanted; because each computer has a different screen layout, and different graphics commands, these calculations are different for each computer.

PLOTTING CIRCLES

We have seen how you can work out the position of a point on the circumference of a circle by referring to the x and y axis, if you know the angle that it makes to the centre of the circle, *and* the radius.

It's also possible to plot the position of the point by working from these pieces of information, and this gives you the key to how you can draw a circle on the screen—or, for that matter, how you can position characters or marks around it as on the compass program.

What you need is a program which tells the computer a centre and a radius for the circle, and how to work out the x and y coordinates for any given angle. Then, if you feed in a whole series of angles, it will be able to plot a whole series of points around the circumference of the circle you have chosen. The tricky bit is how to tell the computer to work out the x and y coordinates, and this is where our old friends SIN and COS come in.

Type in this short program:

```
20 FOR x=Ø TO 2*PI STEP PI/3Ø
3Ø PLOT 128+5Ø*SIN x,88+5Ø*COS x
4Ø NEXT x
```

When you RUN the program, you will see a series of dots which quickly run round to form a circle. There is no reason why the circle should not be continuous, except that it would take a long time to draw.

In each program, except the Spectrum, the computer is first put into a graphics mode. On the Dragon and Tandy, the value of PI is also defined. Then, a FOR ... NEXT loop instructs the computer to move through a series of angles from Ø to 2*PI (Line 3Ø on the Spectrum, Dragon and Tandy, 2Ø on the Commodore, 4Ø on the Acorn)—in other words to form a complete circle. The amount of the STEP here is what speeds the program up, and causes the dotted effect.

In the following Line, the computer is then instructed to PLOT a point for each angle. The position of the point is determined by the formula:

PLOT radius* SIN X, radius* COS X

Remember the circle diagram in **fig 2**, and how these functions determine the coordinates for any given angle. The other numbers added or subtracted in this Line make the formula look much more complicated than this, but the only reason they are there is to set the centre of the circle in relation to the computer's graphics screen. On the Acorn machines this is at 68Ø, 512; on the Commodore it is at 15Ø, 1ØØ; the centre on the Dragon and Tandy is at 127, 95 and the Spectrum's is at 128, 88.

To help you to become familiar with these functions, try altering the centre position—on the Acorn machines you need to change Line 3Ø, which is the centre plus the radius, as well. Or try altering the size of the circle by changing the radius—set at 2ØØ on the Acorn, 8Ø on the Commodore, Dragon and Tandy, and 5Ø on the Spectrum.

There are other numbers, too, which can be changed to alter the circle's appearance. At the first line of the loop, the STEP governs the gap between each dot. The smaller the STEP, the closer to a complete circle would be drawn. If you want more dots in your outline, then decrease the size of the step by making the final number smaller, (or by dividing PI by a larger number).

```
1Ø HIRES Ø,1
2Ø FOR X=Ø TO 2*π STEP.Ø1
3Ø PLOT SIN(X)*8Ø+15Ø,COS(X)*
   8Ø+1ØØ,1
4Ø NEXT X
5Ø PAUSE 1Ø
```

```
2Ø MODE1
3Ø MOVE 68Ø,712
4Ø FOR X=Ø TO 2*PI+.Ø5 STEP .Ø5
5Ø PLOT 69,68Ø+2ØØ*COS(X+PI/2),
   512+2ØØ*SIN(X+PI/2)
6Ø NEXT
```

```
1Ø PMODE 4,1:PCLS:SCREEN1,1
2Ø PI=4*ATN(1)
3Ø FOR X=Ø TO 2*PI STEP PI/45
4Ø PSET(127+8Ø*SIN(X),95-8Ø*
   COS(X),5)
5Ø NEXT X
6Ø GOTO 6Ø
```

Q+A

How large a circle can my computer DRAW?

The size of the largest circle your computer can DRAW is limited, by the size of the screen, to a radius of half the smallest dimension of the screen. The maximum radius for your computer is: Acorn 512, Commodore 100, Vic 88, Dragon and Tandy 95, and Spectrum 88.

For these numbers to be correct, the circle's centre must be that number of pixels into the screen, on both the x and y axes. If the circle is too large the Spectrum gives an 'Integer out of range' error. The other computers draw as much of the circle as they can, although the effect on the Commodore 64 varies.

DRAWING AN ELLIPSE

A much more interesting change can also be made. This will change your circle into an ellipse. To do this, make the number by which you multiply the SIN either larger or smaller than the number by which you multiply the COS.

Multiplying the sine by a larger amount will result in a wide, and short ellipse, while multiplying the cosine by a larger amount will give a taller and narrower ellipse.

The next programs will draw a series of ellipses which together look like a picture of the Globe:

```
1Ø FOR z=Ø TO 5Ø STEP 5
2Ø FOR x=Ø TO 2*PI STEP PI/15
3Ø PLOT 128+z*SIN x,88+5Ø*COS x
4Ø NEXT x
5Ø NEXT z
```

```
1Ø HIRES Ø,1
15 FOR Z=Ø TO 1ØØ STEP 1Ø
2Ø FOR X=Ø TO 2*π STEP.Ø5
3Ø PLOT SIN(X)*Z+15Ø,COS(X)*6Ø
   +1ØØ,1
4Ø NEXT X,Z
5Ø PAUSE 1Ø
```

```
1Ø MODE 1
2Ø MOVE 68Ø,712
3Ø FORZ=Ø TO 2ØØ STEP 4Ø
4Ø FORX=Ø TO 2*PI+.Ø5 STEP .Ø5
5Ø PLOT 69, 68Ø+Z*COS(X+PI/2),
   512+2ØØ*SIN(X+PI/2)
6Ø NEXT:NEXT
```

```
1Ø PMODE4,1:PCLS:SCREEN1,1
2Ø PI=4*ATN(1)
3Ø FOR Z=Ø TO 8Ø STEP5
4Ø FOR X=Ø TO 2*PI STEP PI/45
5Ø PSET(127+Z*SIN(X),95-8Ø*
   COS(X),5)
6Ø NEXT X
7Ø NEXT Z
8Ø GOTO8Ø
```

The Globe programs are a slightly adapted version of the circle program. Instead of drawing one ellipse, a series of them is drawn using a second FOR ... NEXT loop to vary the size of the number by which the x coordinate is multiplied. This makes the computer draw each ellipse a different size.

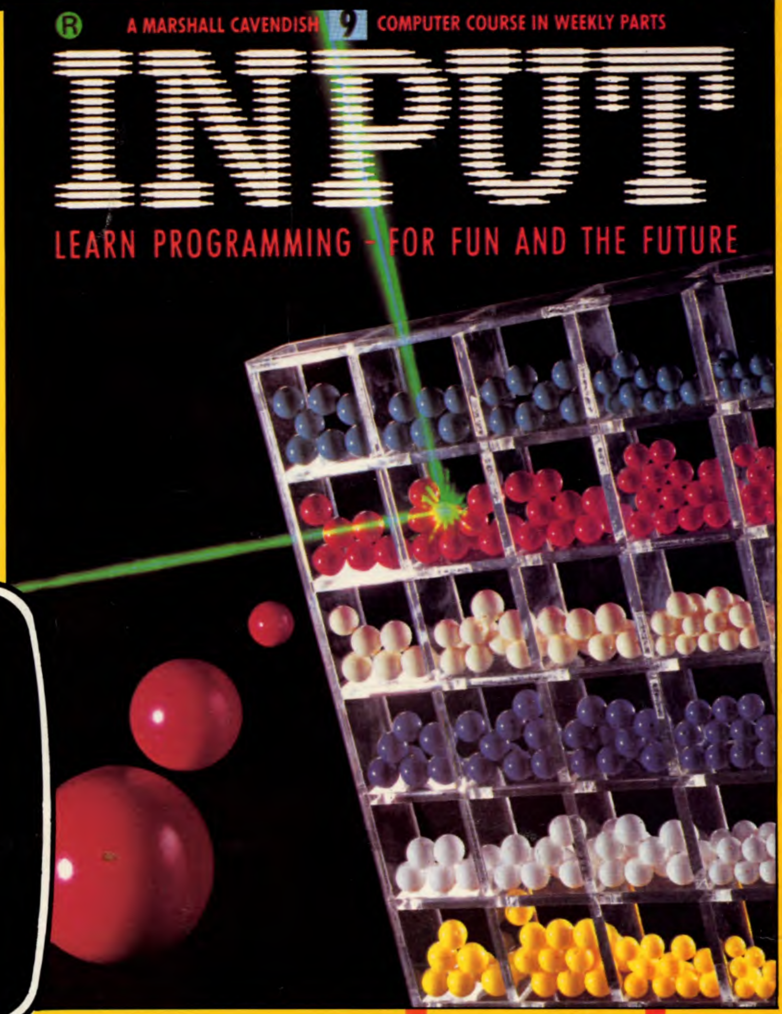
You can make the ellipse 'grow' by any amount you choose by adapting the size and STEP of Z, the control variable.

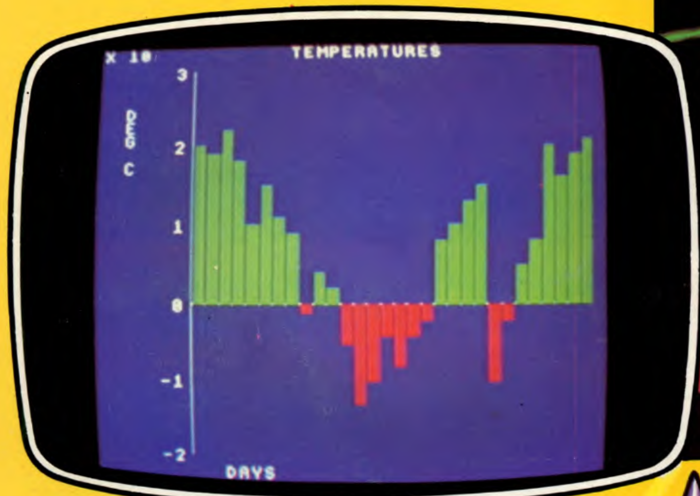
CUMULATIVE INDEX

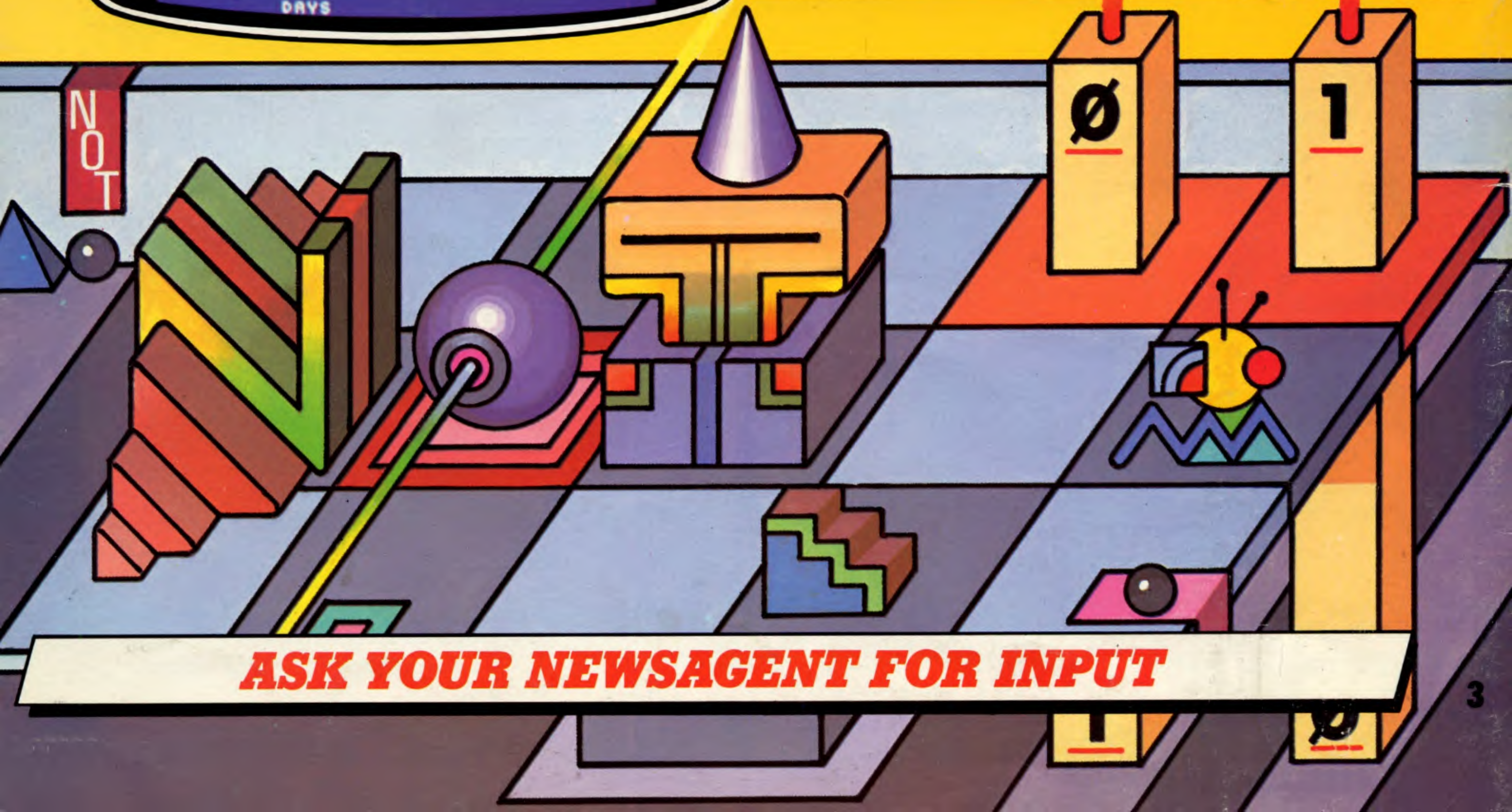
An interim index will be published each week. There will be a complete index in the last issue of *INPUT*.

COMING IN ISSUE 9 ...

- *Display your facts and figures in a professional looking BAR CHART*
- *Find out about ADVENTURE GAMES in the start of a new series*
- *Learn how LOGICAL OPERATORS can extend your decision-making power*
- *Start entering some real MACHINE CODE with a special machine code monitor*
- *If you've a lot of interrelated data you'll need to store it in a 2-D ARRAY*







ASK YOUR NEWSAGENT FOR INPUT